INTERACTIVE PRODUCTION SCHEDULER

by

MURALIDHAR THEEGALA

B.Tech (Mechanical Engineering)

Indian Institute of Technology

Madras, India, 1987

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

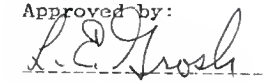
Department of Industrial Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1989

Approved by:



Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

The author wishes to express his sincere gratitude and thanks to Dr. L.E.Grosh, for his guidance, encouragement, and direction. It has been an invaluable experience.

The author also wishes to thank Dr. Doris Grosh, and Dr. Garth Thompson for consenting to be on his advisory panel, and for the guidance during the course of this thesis.

The author also acknowledges the support and help of V.Subramanian and thanks him.

The author dedicates this work to his parents.

CHAPTER 1

INTRODUCTION

1.1 Simulation

Simulation is a technique for solving problems. It can be defined as a procedure in which experiments are performed on a model of a system in order to determine how the system would respond to changes either in its structure or in its environment. It can be much more than just a tool for testing a proposed facility or executing new operating methods.

Simulation is essentially a procedure in which system outputs are defined and analyzed as logical and mathematical functions of system inputs. There are, to be sure, numerous details that complicate simulation methodology--things like random number generators, simulation languages, statistical methods, design of experiments and a host of others. But basically, simulation is nothing more than an efficient way of transforming input to output. Whether we are interested in evaluating a proposed design for a system to be created, assessing the impact of changing a policy, evaluating the feasibility of new products or assessing the impact of additional resources, simulation can give us estimates that describe the efficiency with which inputs are being used.

The potential for applications in this respect is unlimited. Numerous applications exist in manufacturing, including facilities design, production planning, production scheduling, inventory management, quality control, material handling and distribution.

1.2 Interactive Simulation

Broadly speaking, in interactive simulation, the user is allowed to make possible dynamic changes during the course of the simulation. Today it is almost taken for granted, but it was only a few years ago that all simulation was done in a batch mode on a mainframe computer with no graphics. Some interactivity has been provided on the mainframe, but it is usually in the form of input of data and editing models which are then submitted for batch execution. In a similar way graphics became available on the mainframe in a limited fashion. The microcomputer has caused a revolution in the way we output results. Interactivity goes beyond just input or editing of data. It allows the user to observe the program execution and to interact with the system to provide more intelligent results. Summarizing, the interactive simulation has three benefits (Jones, 1985), namely,

* gives the user complete control over the flow of the simulation.
* gives a better understanding of the control

situations involved.

* gives a feel for interaction of the controlled variables and their impact on decision results.

1.3 Simulation Objectives

In reviewing the purposes to which simulations have been put, the objectives of simulation are classified into four general categories (Jones, 1985):

* Improving communication between client and designer-analyst.
* Training, primarily on the operational and managerial levels.
* Predicting and controlling system behavior.
* Design and development of new systems.

At this point, let me emphasize that the objective of my study will be in the simulation training of production management and to facilitate instruction in production scheduling by aiding the visualization of the effects of changes in a production and inventory system.

1.4 Production Management

Production Management is concerned with the organization and control of the production function or, more specifically, it is concerned with the decision-making necessary to ensure that goods are made in accordance with the requisite quality standards, in the requisite

quantities, at the requisite times, and at a minimum cost (Buffa, 1961). Production is one of the central aspects of business in that it has many frontiers with other functions, particularly marketing, personnel, and product research and development. The production function is the key subsystem of a manufacturing enterprise and should therefore figure prominently in any business policy game. Mize (1971) identified the following twelve characteristics of an inventory-production-sales system.

(1) Several finished goods are sold in discrete units.

(2) Periodic demand for each product is a random variable that may follow a trend.

(3) Each product is composed of assemblies and parts.

(4) At least some assemblies are composed of subassemblies and purchased parts.

(5) There are some common components and/or subassemblies among the finished products.

(6) Lead time for a purchased part is a random variable.

(7) Fabrication and assembly operations are performed at work stations.

(8) Different assemblies require processing on some of the same work stations.

(9) The quality of incoming raw materials and of manufactured goods is a random variable.

(10) Processing times at certain work stations are essentially deterministic and at others are random variables.

(11) Machines at work stations experience breakdowns at random intervals.

(12) Repair time of breakdowns is a random variable.

The above points can serve as a checklist for evaluating the degree of realism of a production management system. Such characteristics when incorporated in a simulation model behave like stochastic variables. A stochastic model is one which mimics the random behavior of a system simulated. In contrast, a deterministic model has no random behavior associated with it. This work will basically deal with deterministic models because they give users a better understanding of the system response to minor changes in the input variables.

In order to define the basic data requirements for a production management system, the physical interrelationships that exist among the functional activities of the manufacturing system must be understood logically and comprehensively.

To determine the interrelationships among the functional activities of the firm, a systematic analysis of

the materials, cash flows and the information required to run the manufacturing system has to be carried out. The firm can be considered as a set of four modules: the materials system, the manpower system, the financial system, and the capital equipment system. These four systems form the basic structure of the total operational system and these can be controlled effectively by planning, forecasting, scheduling and inventory control (Partridge and Sculli, 1979).

This simulator is devoted to learning and investigating the scheduling rules which are used to dispatch work at the time of actual production. In previous investigations these rules have been evaluated with respect to selected measures of shop performance, i.e. machine utilization, the amount of work-in-process inventory, the length of the manufacturing cycle times, or the performance against due dates. Many shops, however, maintain an inventory of finished products, and a significant part of their production is directed toward replenishing this inventory. In these cases an important criterion of the scheduling function is the combined performance of the shop and its associated inventory system.

A review of production and inventory management literature reveals little in the way of sophisticated

instructional methods designed to acquaint either college students or real world practioners with a technical basis for future learning in this area of decision-making. My goal in this thesis is to develop such an instructional tool.

1.5 Previous study on simulation of production controls

The work done by Mr. M. Arunachalam, a Kansas State University graduate in 1988, features an interactive next-event simulation package which will allow the user to explore the rules of Optimized Production Technology (explained later in chapter 2) and the theory of constraints. The simulator developed by him allows the user to make decisions and control the flow of the simulation when it is actually in progress. The simulator which runs on an IBM-PC or compatible computer, features a character output in addition to other features such as zoom, future event chain display, and statistics display. It is completely menu-driven and is suitable for use as an instructional tool for students in understanding the dynamics of production control, even if they have very little knowledge about computers.

1.6 Objective of this study

For better control and understanding of the simulation, the user should be able to interact with the

system when the simulation is actually in progress. Also scheduling of the events could be made flexible by specifying the event time in addition to the event type. If one can read in a starting condition, other than the empty shop, many more interesting problems may be investigated. Providing such a facility in the simulator will allow the user to save the current simulation state on the disk and to run the simulation at a later stage from that point. Using the input file as a preload device allows one to do external analysis using a spreadsheet, say, and postulate scheduling alternatives which may be evaluated by the simulator. Finally it is interesting to contrast the simulator with the scheduling of a real-life plant whose production processes are greatly more complicated than the existing ones. However, the number of constraints, even in real-life, is always small, it is just that the number of production schedules that can be planned with varying combinations of the constraints is enormous. Developing the above extensions to the existing simulator is my objective for this thesis.

Specifically these extensions are listed as follows:

1. Allow events to be scheduled at a future time.
2. Record progress in a file by "WRITE".
3. Re-establish the simulation by reading in a file created by "WRITE" or by a word processor.

4. Correct system problems which were found in the previous simulator:
    a) Load
    b) Move
    c) Transfer lot problems caused by the master production schedule, and
    d) Print hang up.
5. State and use a master production schedule.
6. Create a measurement of completed inventory and work in progress.
7. Create multiple input files to illustrate their use.
8. Begin the simulation at time now = 0 by the RESUME key.
9. Finally, write a comprehensive User's Manual.

CHAPTER 2

OPTIMIZED PRODUCTION TECHNOLOGY

2.1 Introduction

For the past few years, the term optimized production technology has been heard more and more in production and inventory control circles. It is a new philosophy of Production/Operations Management advocated in "The Theory of Constraints" or "OPT" [Goldratt and Fox, 1983].

The way that production is planned and scheduled can be critical to the profitability and survival of manufacturing companies. Understanding the correct principles of scheduling is important in selecting between alternative approaches such as OPT, MRP and Just-in-Time. Like just-in-time, OPT attacks waste in the factory, but more efficiently. OPT focuses on the critical resources, the ones that really control output. By directing management to focus its energies on bottlenecks, it succeeds in maximizing throughput and making more money. Furthermore, OPT's unique scheduling system makes it a powerful simulation tool that permits the user the luxury of measuring the effects of planned improvements before one dollar is spent on them. This chapter will discuss the conceptual development of the theory and the operational rules based on it. These rules will then be emphasized

with an interactive computer program which can accept user/management decisions and act accordingly.

2.2 OPT Scheduling Rules

The way OPT looks at manufacturing is based on the following ten scheduling rules [Fox, 1983]:

(1) Balance flow, not capacity.

(2) The level of utilization of a non-bottleneck is determined not by its own potential, but by some other constraint in the system.

(3) Utilization and activation of a resource are not synonymous.

(4) An hour lost at a bottleneck is an hour lost for the total system.

(5) An hour saved at a non-bottleneck is just a mirage.

(6) Bottlenecks govern both throughput and inventory in the system.

(7) The transfer batch may not, and many times should not, be equal to the process batch.

(8) The process batch should be variable, not fixed.

(9) Schedules should be established by looking at all of the constraints simultaneously. Lead times are the result of a schedule and cannot be predetermined.

(10) The sum of the local optimums is not equal to the global optimum.

The rules 1, and 6 through 10 could be illustrated on

our fully developed simulator by some standard problems. Actually these may also become obvious when one keeps using this simulator more often.

In addition, there are three criteria that the schedules must meet:

1. Move the organization toward the goal of manufacturing : MAKING MONEY
2. Be realistic
3. Be immune to disruption

2.3 How OPT works?

This section of the chapter will show how the OPT system works and utilizes these rules [Fox, 1984].

The first step is to construct a model of the manufacturing facility to be scheduled. The model is essentially a network of how the various manufacturing resources (people, machines, tools, etc.), customer orders, products and raw materials are linked together. The creation of the initial model of the manufacturing facility is accomplished using data generally available on a company's computer system. Existing data files containing bills of materials, routings, market considerations (sales forecasts and firm orders), inventories and work center data (setup and processing times) are fed into the OPT system.

Once this model has been constructed, the next task is

to determine where the constraints or bottlenecks are in the system [Meleton, 1986]. In OPT terminology, any resource whose capacity is less than or equal to market demands is called a bottleneck. This is accomplished through a module in OPT called "Serve". "Serve" is OPT's enhanced version of an MRP system. The product network of OPT is combined with what is in essence the MPS, and this is fed to a routine that identifies the bottleneck resources; basically, this is a rough cut capacity planning routine that achieves many of the benefits of capacity requirements planning. A pass through this network gives a capacity estimation for each work center, and a gross to net calculation can be made at each part of the routing step to see if further capacities are required. The resultant capacity needs, when divided by the number of weeks in the planning horizon, give the average capacity requirements for each resource. When this is divided by the resource capacity, the result is average expected load. The average loads are then sorted in descending order, and the most heavily loaded are studied. Are the data correct? Are the time standards accurate? Can the capacity be easily increased? Can an alternate routing for some items be used? Any changes based on these questions dictate another run to find the true bottleneck resources. This way of identifying the bottlenecks is called infinite loading.

Once a bottleneck is identified, the model is divided into two parts: the critical resources and the non-critical resources. The critical resource portion is our bottleneck operation. This dividing of the model is done by another module in the OPT software called "Split". The critical resource portion of the model will be scheduled using a module called the "Brain of OPT". The "Brain of OPT" not only schedules production but also determines the transfer batch and process batch sizes at each operation. It attempts to maximize the throughput at the bottleneck operation while at the same time maintaining a synchronous flow of parts to assure the correct mix of parts is being produced.

We can see that OPT clearly meets our three criteria for correct schedules:

* Meeting our goals of making money: the OPT schedules generate the maximum throughput with the least inventory for a given set of operating expenses.
* Realism: detailed schedules are generated that can be analyzed by first line supervision to assure they are doable.
* Immune to Disruption: Safety stock and safety capacity are employed at strategic points to prevent any disruption in the throughput of the total system.

2.4 OPT - A Productivity Improvement Tool

If OPT is able to generate schedules that meet the three criteria of good scheduling, then it should also be a powerful simulation or "what if" tool. It is useful to think of OPT as a system that produces the best practical schedule to meet marketing demands under these three types of constraints:

* Materials: What is our current inventory position on raw materials?
* Capacity: Capacity is a variable that can be changed but at any point in time our capacity is fixed.
* Policy: These constraints can be thought of as management parameters or knobs on the OPT tool to meet their specific needs.

2.5 Use of OPT in controlling the Simulator

It is suggested that the user understand the philosophy of OPT before using the simulator. Judicious use of these rules in controlling the flow of the simulation will help the user understand the dynamics of production control and allow him to realize the benefits of using these rules to achieve the goal of a manufacturing organization. The organization of the simulator is described in Chapter 3 of this study. A comprehensive user's manual for the simulator follows in Chapter 4.

CHAPTER 3

ORGANIZATION OF THE SIMULATOR

3.1 Introduction

This chapter will discuss the basic data structures that have been changed since the development of the last simulator. It will also give a brief insight into the organization of the program. Since this work is an extension to the previous study, for a better understanding of the program and for proper definitions, the previous work needs to be referred to quite often.

Certain changes, though not directly related to the objectives of this study as mentioned in Chapter 1, were of significant importance to the overall development of the simulator. Sections 3.2 and 3.7 cover these changes in detail.

3.2 Calendar Data Structure

The calendar is an important data structure and forms the backbone for the program. It contains all the scheduled future event types. The first field in the calendar is the modifier which decides if that particular event is mandatory. The critical modification has been in giving a symbol to the REG_MODIFIER. The symbol is '@' instead of a <space> which helps in recognizing the end of file for reading in the starting conditions other than the empty shop, as explained later in section 3.5.2.

3.3 Other Modified Data Structure in the Program

In addition to the calendar data structure discussed earlier, the program uses the following modified data structure.

3.3.1 Product Structure

The structure for a product is defined as follows in 'opt.h':

Before Modification:

```
#define PRODUCT struct products

PRODUCT{
    int prod_no;        /* Product number */
    int price;          /* Product price */
    int qnty_produced;  /* Quantity produced */
    int row;            /* Screen row */
    int col;            /* Screen column */
    PRODUCT *next;      /* Next product */
};
```

After Modification:

```
#define PRODUCT struct products

PRODUCT{
    int prod_no;        /* Product number */
    int price;          /* Product price */
    int qnty_produced;  /* Quantity produced */
    int limit;          /* Master Production Schedule or
                           quantity sold per week */
    int row;            /* Screen row */
    int col;            /* Screen column */
    PRODUCT *next;      /* Next product */
};
```

The variable 'limit' contains the scheduled quantity that can be sold per week. When the simulation is in progress, this information is displayed in the overview

window at the location "Products to be sold".

3.4 Flexibility in Scheduling

The way that production is scheduled can be critical to the profitability and survival of manufacturing companies. The OPT way of thinking tells us that we must look at all the constraints simultaneously. If a machine or raw material is not a constraint, then it is not important in setting the master schedule. There are a number of other different constraints that need to be considered. The most obvious one is the market requirements. There are clearly a limit on what we need to produce and when we produce it. The original simulator worked on the assumption that whatever was produced could be sold immediately. But in reality this is not true. So the following change was made in the modified simulator.

3.4.1 Master Production Schedule

The simulation begins after reading the starting conditions from an input file. At this stage the limit in the product structure represents the master production schedule, the quantity which can be sold per week. This limit is automatically reset at the end of the week. This modification is done in the function called "reset_quantity".

3.4.2 Purchase_Rawmat

This procedure purchases raw material 'raw_mat_no' (event variable 1) for part_type (event variable 2) in 'qnty' (event variable 3). Purchase events are scheduled by a function called "check_inventory" if the user chooses automatic purchase. In such a case the event is scheduled at the next available minute. In the modified version of the simulator, when the user wants to purchase some raw material, he needs to specify an event time in the form of week, day, hour and minute. This information is used to convert the time into minutes by the function "string_to_time". At this point, the time was checked for input validation. Only when the input time was greater than the present time of the simulation, did the procedure execute. In this way the user can control the flexibility in scheduling an event. This change has been incorporated in the function called "user_purchase".

3.4.3 Setup_machine

This procedure will set up the machine specified by 'mach_no' (event variable 1) in department 'dept_no' (event variable 2) for part type 'repeat, part_type' (event variables 3 & 4). In the manual operation, even this event can have flexible scheduling by specifying the event time in the form of week, day, hour and minute. This change has been incorporated in the function called "change".

At the later stage of the program development, a serious bug was detected in this routine. First of all this procedure was accepting wrong input, i.e. even though certain part types do not go through a particular department, when the user by mistake scheduled to set up a machine for any of those part types, the procedure did so. The program detected wrong part information immediately at the next minute and terminated the simulation abruptly. The whole problem was in the way these part types were compared with the existing part types in the part information data structure. Actually they are represented in the program by character variables, but the original procedure was doing a string comparison on them. This may appear to be a trivial problem on paper, but avoiding this by proper modification, did set the setup procedure right.

3.4.4 Load_machine

The load event can take one of the two forms shown below:

Load 'mach_no' in 'dept_no' setup for 'repeat, part_type'

(or)

Load free machine in 'dept_no' setup for 'repeat, part_type'

Originally in the second case, the function called "check_free_machs" was used to schedule the load events in automatic mode. But there was a bug in the routine. It

was just checking for only one machine in a department. Let me give an example. The load events are scheduled basically when a machine has been set up for a particular part type and is sitting idle and when there is at least one part in the "pre" area. The procedure just checked for the first machine in each department. This was corrected by looping around all the machines in the department. In addition, the function called "user_load" has been modified to incorporate the flexible scheduling.

3.4.5 Move_part

The move event is of the form:

Move 'repeat, part_type' from 'dept_no' in 'qnty'

The destination for the part is implicitly mentioned. In the original simulator only the automatic move was working fine. There was no interaction for the user to move a part in the manual mode. This has been provided in the function called "user_move". The procedure finds the destination for the part by searching the routing matrix. If the destination is another department, it moves the part to that department and updates the expense by adding material handling cost to it. If the destination is stores, which is represented as the 0th department in the routing matrix as explained later in Chapter 5, it moves the part to the stores and updates the cash by adding the product price to it and updates expense by adding the

material handling cost to it. But at the same time, the part cannot be moved if we exceed the limit as dictated by the master production schedule. In that case the material remains at the "post" area of the last department and the on_hand inventory is updated. The on_hand inventory is explained in the next section.

3.4.6 Unload_machine

In the original work, since there was no user move, this procedure unload_machine automatically generated a move event in the next available minute. But when the simulation was running with manual option and with the user_move feature provided, such an event generation was no longer justified. Also once the part is processed through the last department, it had to be moved either to finished goods inventory (i.e. "on hand") or to the area where it can be sold directly. These modifications were incorporated in the function called "unload_machine".

3.5 Statistics

The statistics window consists of three reports: sales, financial and activation. A sample layout of the statistics window is shown in Figure 3.1.

The sales report shows the number of pieces of each part produced from the beginning of the simulation to any moment. It also shows the on_hand quantity for each product type. When the parts are being produced and we

```
WEEK: 00  DAY: 0              STATISTICS                    TIME:  5:10

      FINANCIAL REPORT                      ACTIVATION REPORT

        Starting Cash   25000                Mach#  Prod%  Setup%  Idle%
            Cash Now    22576   Dept 1         1      22      77      1
                                               2       0     100      0
     Raw Mat Expenses    2050                  3      43      51      6
Mat Handling Expenses     71   Dept 2         1      52      41      7
    Other Op Expenses      0                  2       0       0    100
Work_in_process Value    303                  3       0       0    100
   Revenue from Sales      0   Dept 3         1      32      67      1
        Net Cash Flow     +0                  2       0      45     55
                                              3       1      19     80
         SALES REPORT         Dept 4         1       0      30     70
           Sold  On Hand  WIP               2       0       4     96
Product A:   0       0     20               3       0       0    100
Product B:   0       0     25   Dept 5      1      12      41     47
Product C:   0       0      5               2       0       0    100
Product D:   0       0     25               3       0       0    100
Product E:   0       0     25

F1Overview F2Zoom F3Calendar  Resume  Print  Quit  <+/->Pace
```

Figure 3.1 Sample layout of the Statistics Window

exceed the quantity that could be sold per week, the on_hand or the finished goods inventory keeps building up. This is a new feature which has been added to the simulator. The functions "init_on_hand" and "update_on_hand" do this job for the sales report. Also the sales report shows the Work_in_process (WIP) inventory for each product type. The WIP is calculated by subtracting the on_hand inventory from the value obtained by adding up the quantity from both "pre" and "post" areas of all the departments for each part type. The function "update_stat_inv_info" does this job for the statistics window.

The financial report consists of starting cash, cash now, raw material expenses, material handling expenses, other operational expenses, work in process value, revenue from sales, and net cash flow.

The starting cash is recorded to be the cash now at the beginning of each week. The cash now reflects the cash on hand at any moment. Raw material expense is calculated and updated by "purchase_rawmat" function. Material handling expense is updated by the "move_part" procedure. The work in process value is calculated and updated by "update_inv_finance" procedure. The formula used for calculating WIP is:

```
WIP value = summation of (inventory[i][j] * (raw_cost[i] +
            j * (prod_cost[i] - raw_cost[i])/no_of_opns[i])
```

```
for  j = 1 to # of operations, and
for  i = 1 to # of part types, where
inventory[i][j] = sum of pre- and post- areas at department
                  j for part type i;
raw_cost[i] = raw material cost for part type i;
prod_cost[i] = product price for part type i; and
no_of_opns[i] = total number of operations for part type i.
```

Other operational expense and net cash flow are calculated at the end of each week.

The activation report shows the percentage of total time spent in setup, production, and idle states. The event procedures which change the state of the machines- namely setup, load, and unload- mark the time of the state change and update the appropriate variable 'time_idle', 'time_setup', or 'time_busy' according to what the machine was before the state change.

3.6 Recording Simulation State

3.6.1 Introduction

The simulation is started from an input file which has values for all the variables which define the problem. This is basically simulating an empty shop situation. But if one can read in a starting condition, other than that of the empty shop, other more interesting problems may be investigated. It would also be an invaluable addition if the user is able to save the calendar and other related simulation data on the disk and continue from that point at another time. This would save a lot of time if the user wants to run the same problem again. Another reason why

the user might want to save the calendar at any time would be to try two different alternatives from that particular point and evaluate them by observing the system performance.

Providing such a facility in the simulator has been a major task for my research. The next two sections explain in detail on this feature.

3.6.2 Writing Simulation State

At any point of the simulation, writing the simulation state to a file is achieved through a two step menu process. The 'Schedule' option from the overview window menu list will bring up the user scheduled events menu list. Then the 'Write' option from this menu will prompt for a filename to be used for writing to the disk. The prompt 'finished writing' will signify the end of saving the simulation state.

The most important variables that change during the course of the simulation are the global variables. Most of these define the initial state also. So these variables are written first to the file:

'cash' for starting cash;

'expense' for all raw material and operating expenses;

'n_depts' for defining the number of departments;

'n_ptypes' for defining the number of part types;

'max_opns' for defining the maximum number of

operations for any part type;

'Routing matrix' to designate the routes for each part type;

'Department structure' consisting of 'dept_no', 'nmachines', 'nmach_in_use', 'row' and 'col';

'Machine structure in each department' representing the information 'row', 'col', 'mach_no', 'repeat', 'setup for', 'mstate', 'user_stopped', 'state_change_time', 'idle', 'setup' and 'busy';

'Part information in each department' defining 'dept_no', 'row', 'col', 'repeat', 'ptype', 'ptype_in_char', 'pre' and 'post' quantity;

'Raw material structure' containing 'raw_mat_no', 'cost', 'row' and 'col';

'Product structure' containing 'prod_no', 'price', 'qnty_produced', 'limit', 'row' and 'col';

'Setup times' for the n_ptypes in n_depts;

'Machining times' for the n_ptypes in n_depts;

'Parttype : raw material relationship';

'time_now' for present simulation time in minutes in the form of week_now, day_now, hr_now and min_now;

'auto_purchase', 'auto_setup' and 'auto_move' for true or false;

minimum inventory, purchase lot-size, transfer batch, material handling cost and other operational expenses - these are important variables because they can be changed

later in the program.

'Statistics structure';

And finally in the end the calendar listing of the events. A lot of care has been taken in writing this simulation state with well formatted statements so that the user can understand them.

3.6.3 Reading Simulation State

Once a simulation state has been written to a file on the disk, we now have two ways of starting the simulation next time. The input file 'in.sim' starts the simulation from an empty shop situation. Any file with '.out' as an extension which has been written previously to the disk, can be used to start the simulation from other than the empty shop condition. The sequence in which the program reads the variables can be understood by seeing the listing for the function 'read_sim_state' in the file 'input.c'. The strings describing the corresponding numbers are written as a guide for the user and are omitted by the program during input.

The very important variables that decide the length of the department and machine structure are read through the number of departments, number of part types and maximum number of operations. The machine structures and the department structures can be initialized without any problem. But once it comes to the part information data,

there is no idea as to how many part types go through each department. So scanning the input file for part information had to be handled delicately and is achieved by reading in the 'dept_no' field first.

Similarly the reliability of the calendar data structure is very important as it is the calendar containing the future event list that decides the flow of simulation. The calendar list is read in the end since there is no prior knowledge of how many events may be in the calendar at any one time. The modifier field is scanned first and the calendar structure is continually initialized till the end of file condition terminates the input. If the modifier had been <space> instead of any character then it would be difficult to recognize the end of file. This is the explanation for the reasoning in section 3.2.

3.7 Memory Model of the Simulator

A greater control can be gained over how the program uses memory by specifying the memory model for the program. The original simulator was compiled in the small memory model. The small-model option tells the compiler to create a program that occupies the two default segments: one for code and one for data. Small-model programs are typically C programs that are short or have a limited purpose. Since code and data for these programs are each limited to 64K,

the total size of a small model program can never exceed 128K. Most programs fit easily into this model.

With the extensions that were incorporated in the simulator, the program now has more than 64K of code, so it was switched from small to medium-model option to compile the program. The medium-model option provides a single segment for program data, and multiple segments for program code. Each source module is given its own code segment.

Medium-model programs are typically C programs that have a large number of program statements (more than 64K of code), but relatively small amount of data (less than 64K). Program code can occupy any amount of space and is given as many segments as needed; total program data cannot be greater than 64K. The medium model provides a useful trade off between speed and space, since most programs refer more frequently to data items than to code as we do.

3.8 Summary of the Modified Changes

C is used to program our simulator. It is a general purpose programming language which features economy of expression, modern control flow and data structures and a rich set of operators. It also provides maximum support for modular programming, which speeds program development and simplifies maintenance, since each module can be designed and tested independently. A modular program is one that is constructed from a set of small, independent

functions, each of which does a single clearly defined job.

The program changes which have been added to this simulator are very modular and can be used for further program development. These changes are of two types:

(i) modifications to the existing functions, and

(ii) addition of new functions.

The first group of changes is:

* 'user_schedule' in file calendar.c
* 'change' in file calendar.c
* 'read_data' in file input.c
* 'main' in file opt.c
* 'move_part' in file process.c
* 'unload_machine' in file process.c
* 'update_prod_info' in file process.c
* 'check_free_machines' in file process.c
* 'set_screen' in file setscr.c
* 'draw_menu' in file setscr.c
* 'draw_products' in file setscr.c
* 'init_stat' in file stat.c
* 'draw_stat_screen' in file stat.c
* 'string_to_time' in file s_conv.c
* 'user_purchase' in file utils.c
* 'user_setup' in file utils.c
* 'zoom' in file zoom.c

The new functions added are:

* 'write_cal_list' in file calendar.c

(Function to write the simulation state of the calendar list)

* 'read_cal_list' in file calendar.c

(Function to read the new simulation state of the calendar list)

* 'read_sim_state' in file input.c

(Function to read previously saved simulation state data from the file specified by the user)

* 'update_inv_finance' in file process.c

(Function to update the inventory costs in statistics window)

* 'init_on_hand' in file process.c

(Function to initialize the on-hand inventory in statistics window)

* 'update_on_hand' in file process.c

(Function to update the on-hand or finished goods inventory in statistics window)

* 'update_stat_inv_info' in file process.c

(Function to update inventory information on the statistics window)

* 'reset_quantity' in file setscr.c

(Function to reset the quantity produced for the next week)

* 'draw_initial_scr' in file setscr.c

(Function to paint the initial screens when started from a

saved simulation state)

* 'write_sim_state' in file utils.c

(Function to get the user input for writing simulation state to a file)

* 'user_move' in file utils.c

(Function to get the user input for scheduling move events)

CHAPTER 4

USER'S MANUAL

4.1 Introduction

This user's manual will give some insight into the features of the simulator. The manual also features a menu map to help in efficient use of the simulator's capabilities.

4.2 Hardware Requirements

The simulator is suitable for use on IBM-PC and compatibles with a Color Graphics Adapter (CGA) or an Enhanced Graphics Adapter (EGA). The executable version of the simulator is on a 5 1/4" double sided double density floppy disk and hence requires a floppy disk drive.

4.3 Getting Started

The simulator is started by inserting the disk in a disk drive and typing 'OPT' followed by <Enter>. The program checks the hardware for the availability of Color/Graphics adapter. It then prompts for two options: whether to start the simulation from the beginning or to start from a previously saved simulation state. Once the option is given, it prompts for the input filename. A sample input file 'in.sim' is provided on the diskette for first option. If the user selects the second option, then he can type in any filename which has '.out' extension.

The next two sections cover these two options in detail.

4.3.1 Initial Data Input Through Infile

The input file 'in.sim' is listed in Appendix B. It is also available on the disk with an executable version of the simulator. The simulator can accept any file as input as long as it is in a specific format which can be read by the program to understand the input. It is suggested that the user make a copy of 'in.sim' before attempting to modify it. It can be edited by any word processor in nondocument mode. The strings describing the corresponding numbers are in the input file only as a guide for the user and are omitted by the program during input. The sequence in which the program reads the variables can be understood by seeing the listing for the function 'read_data' in the file input.c in Appendix A. Although the strings are omitted by the program, they cannot be removed nor can more strings be added in the input file. The variables listed in the input file in.sim also form a subset of those listed in any .out files. So in the next section a detailed explanation of these variables from a file with .out extension are covered for a better understanding to the user.

4.3.2 Previously Saved Simulation Data Input

The input file 'final.out' is listed in Appendix B. The sequence in which the program reads the variables can

be understood by seeing the listing for the function 'write_sim_state' in the file input.c in Appendix A. A brief explanation for each of these variables follows:

4.3.2.1 Global Variables

cash: cash now available
expense: expenses incurred
n_depts: number of departments
n_ptypes: number of part types
max_ops: maximum number of processing stations or departments
pace: the pace of the simulation

All these are global variables. They should be initialized for the program to run properly.

4.3.2.2 Routing

A typical routing matrix has the form:

```
1 3 5 2 0 0 0
3 4 5 2 1 2 0
1 5 2 5 3 4 0
1 3 5 4 0 0 0
2 5 4 3 1 2 0
```

The five rows relate to the 5 part types and the numbers in the first 6 columns represent department numbers or processing stations. The additional column in the matrix with all 0's signify the completion of the processing when the part types are ready to be sold or to be placed at finished goods inventory.

4.3.2.3 More Global Variables

routing_row: screen row position for routing display
routing_col: screen column position for routing display
raw_mat-row: screen row position for the raw material

display
raw_mat_col: screen column position for the raw material display
prod_row: screen row position for the product display
prod_col: screen column position for the product display
n_raw_mat: number of raw materials
n_products: number of products

4.3.2.4 Department Structure

dept_no: department number
nmachines: number of machines
nmach_in_use: number of machines in use
row: physical screen row location for the department
col: physical screen column location for the department

4.3.2.5 Machine Structure

row: screen row location for the machine
col: screen column location for the machine
mach_no: machine number
repeat: repeat for part type
setup_for: part type setup for
mstate: machine state

There are three states for any machine. They are:

state	value of 'mstate'
Idle	0
Setup	1
Busy	2

usr_stopped: has any event been stopped by the user
value of 0 means False
value of 1 means True

st_chg_time: contains the marked time of state change for that particular machine

idle: contains the cumulative times from the beginning
setup: of the simulation in these respective states for
busy: the machine which changed state

4.3.2.6 Part Information Structure

dept_no: department number
row: screen row location for part type
col: screen column location for part type
repeat: repeat for part type
ptype: part types
ptype_in_char: part type in character

For part types 1, 2, 3, 4 and 5, the corresponding part types in character are A, B, C, D and E.

pre: contents of the "pre" area
post: contents of the "post" area

The department data structure in itself contains the machine structure as well as part information structure. Figure 4.1 gives the overview of the department data structure.

4.3.2.7 Raw Material Structure

raw_mat_no: raw material number
cost: raw material cost
row: screen row
col: screen column

4.3.2.8 Product Structure

prod_no: product number
price: product price
qnty_produced: quantity produced
limit: master production schedule
row: screen row
col: screen column

4.3.2.9 Setup Times

repeat= 0

240	120	320	160	180
80	120	220	0	130
60	210	100	225	130
0	110	120	115	220
60	220	110	135	130

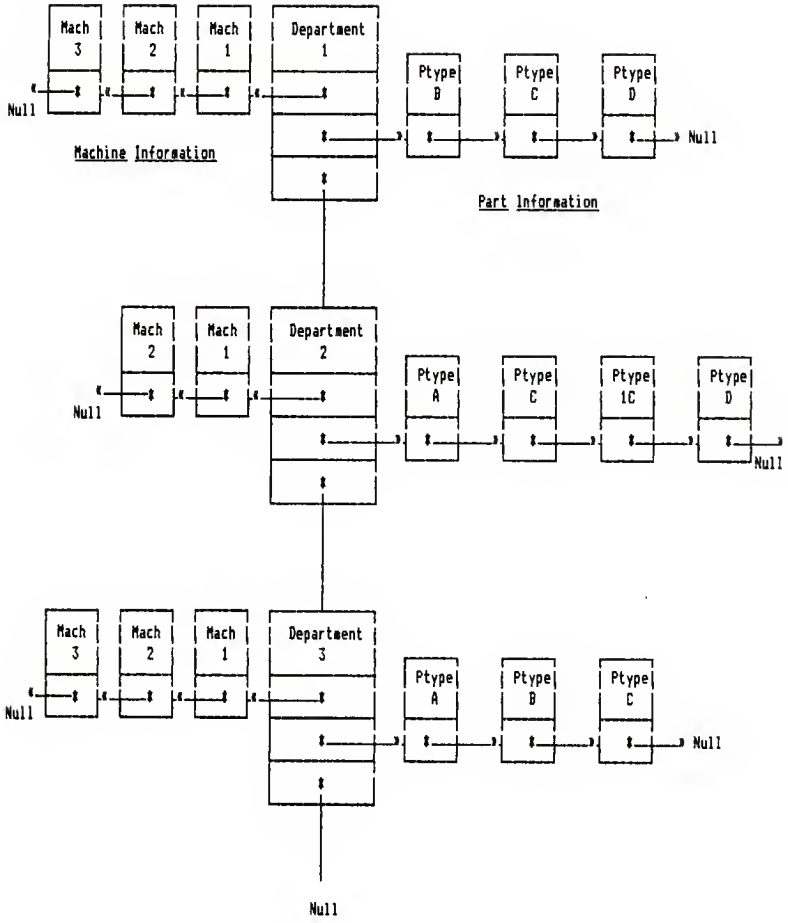


Figure 4.1 Overview of the Department Data Structure

Setup times are represented by a 2-Dimensional 5x5 matrix. Rows signify departments and columns are the part types.

```
repeat= 1
  0    0    0    0    0
  0  125    0    0  165
  0    0    0    0    0
  0    0    0    0    0
115    0  100    0    0
```

If any of the part types has a routing that goes through a department twice, then it is supposed to have a repeat.

4.3.2.10 Machining Times

A similar kind of explanation holds true for machining times as for setup times.

4.3.2.11 Part Type-Raw Material Relationship

This information represents a one-to-one relation between part type and raw material.

4.3.2.12 Some More Global Variables

time_now: simulation time in overall minutes
week_now: week of simulation
day_now: day of simulation
hr_now: hour of simulation
min_now: minute of simulation

deleted: Is an event deleted by the user?
0 for No and 1 for Yes

user_specified: Has an event been specified by the user?
0 for No and 1 for Yes

auto_purchase: Is purchase automatic?
0 for No and 1 for Yes

auto_setup: Is setup automatic?
0 for No and 1 for Yes

auto_move: Is move transfer automatic?
0 for No and 1 for Yes

zoom_row: screen row for zooming a department
zoom_col: screen column for zooming a department

The following global variables can be changed during the simulation also:

min_inv: minimum inventory before automatic purchase is done
purchase_lot_size: default automatic purchase lot size
transfer_batch: transfer batch quantity for material handling
mat_handling_cost: material handling cost of a transfer batch
other_op_expense: other weekly operating expenditure

4.3.2.13 Statistics Structure

row: screen row for finance
col: screen column for finance
start_cash: starting cash for a week
cash_now: cash at this moment
sales_revenue: revenue from sales
raw_mat_exp: total raw material expenses
mat_hand_exp: total material handling expenses
other_op_exp: total operating expenses
net_cash_flow: cash flow for previous week

4.3.2.14 Additional Global Variables

The following global variables represent the information about the first event in the calendar list.

event_time: time of the event
week: week of the event
day: day of the event
hr: hour of the event
min: minute of the event

The event variables event_var1, event_var2, event_var3 and event_var4 are explained in section 4.3.2.15. The

event_type and the modifier are also explained in that section.

4.3.2.15 Calendar Structure

The first entry in the calendar structure, 'modifier', is a character which decides if that particular event is mandatory. If the event is mandatory, it cannot be deleted by the user with the calendar edit function. The operation of the calendar edit function is described in section 4.7.2. A summary of the modifiers follows:

Modifier	Symbol	Comments
CLOCK_MODIFIER	'$'	Mandatory
UNLOAD_MODIFIER	'$'	Mandatory
REG_MODIFIER	'@'	Can be deleted
DELETED	'*'	Not processed
USER_SPECIFIED	'U'	Entered by the user

The next entry 'time' is the event time. The variables week, day, hrs and mins which are the next four entries in the structure are used to store the time in terms of weeks, days, hours and minutes respectively. The next field in the calendar structure is event_function_ptr which is merely the event type. There are six different event types that will change the state of the shop. Associated with these event types, there are four different event variables explained below:

Event	Event Variables
clock_tick	a: dummy b: dummy c: dummy d: dummy
purchase	a: raw material number b: part type c: quantity d: dummy
setup	a: machine number b: department number c: repeat d: part type
load	a: machine number b: department number c: repeat d: part type
unload	a: machine number b: department number c: dummy d: dummy
move	a: repeat b: part type c: department number d: quantity

4.4 Initial Options

The initial display screen consists of the zoomed window showing the first department and the variables that can be changed during the simulation. Of the variables displayed on the screen, two - 'hours per day' and 'days per week' - can be changed only before starting the simulation. A sample layout of this zoom window is shown in Figure 4.2.

The program first prompts the user if purchase events

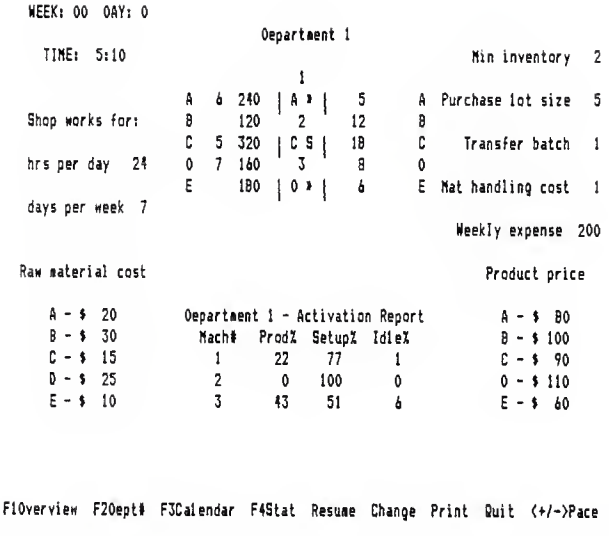


Figure 4.2 Sample layout of the Zoom Window

are to be scheduled automatically. A choice of 'yes' automatically replenishes inventory when the simulation is in progress. A choice of 'no' requires the user to schedule purchase events.

The second prompt is if the setups are to be done automatically. If the user's choice is 'yes', setups are done automatically. If the choice is 'no', the user should schedule setup events to start the simulation.

The next question is if the material handling should be automatic. If the user's choice is 'yes', material handling is done automatically in the pre-specified transfer batch size. If the choice is 'no', the user should schedule move events.

The above modes can be toggled between 'Auto' and 'Manual' by the 'change_mode' function which is invoked by pressing 'ALT_M' from the overview window any time during the simulation.

The program then asks the user if any of the variables displayed on the screen needs change. If the user chooses to alter some variables, the program prompts for a new value for the selected variable. Instructions on how to change the variables are provided in section 4.7.1.2. The display is switched to the overview window after accepting the changes.

If the user chooses not to change any variables, the display is immediately switched to the overview window. At

present the simulation is in the frozen state. When the user presses 'Resume', the simulation clock starts and if the user has chosen automatic setup, they are begun. Since the setup event is the primary event which can start the simulation, the user should schedule setup events to start the simulation if the program is in manual setup mode.

4.5 The Overview Window

This window has information about all the departments and the routing for all the part types displayed. It contains information about the raw material cost, products to be sold or master production schedule, cash on hand and expense from the beginning of the current week. It features a clock in 24-hour format. Also displayed are the number of weeks from the beginning of the simulation and the day of the current week. The window has a menu bar in the bottom of the screen which shows the different options available to interact with the program. An option is selected by pressing the highlighted key for the desired function. The features of the overview window are shown in Figure 4.3. The menu bar and the options are discussed further in section 4.7.

4.6 Description of Symbols in the Overview Window

There are some symbols displayed on the machine image and to the sides of the work-in-process areas which need

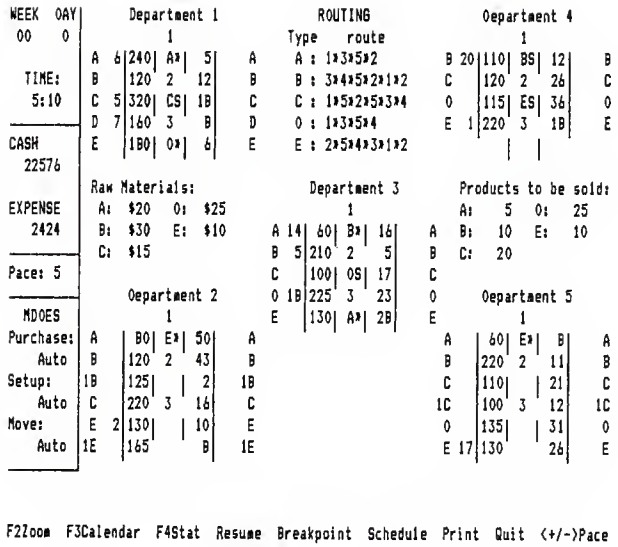


Figure 4.3 Sample layout of the Overview Window

explanation. A blinking 'S' on the machine image means that the machine is being set up for the part type printed on it. A blinking '>' means that the machine is processing the part type printed on it. A non-blinking '+' sign on the machine means that the machine is idle, and is set up for the part type printed on it. The work-in-process areas have the part types displayed on the either side of them. The number beside the part type is the repetition count for that part type in that department. This is decided from the routing matrix which is read from the input file. For example, if the part type 'B' is to be processed by department 3 twice, then the work-in-process area for department 3 will have two areas for the part type 'B', one labeled 'B' and the other labeled '1B'. This is to avoid mixing up the parts of the same type in different stages of processing.

4.7 Menu Functions

The functions in the menu are of two major categories. They are functions which affect the simulation and those which don't. The following sections give brief descriptions for each of the menu functions.

4.7.1 Functions Which Affect The Simulation

The following sections will discuss the functions which affect the simulation.

4.7.1.1 Break Point

The break point is used to freeze the simulation at user specified times so the user can analyze the system state at those times before resuming. The break point can be changed by the user by pressing the 'B' key when the simulation is in progress or when it is frozen. The user is prompted for a new break point. The old value is displayed in the prompt line. The break point is entered in minutes. There is a mandatory break at the end of each day and so this function will not accept a value greater than the number of minutes in a day.

4.7.1.2 Change

This function is invoked by pressing the 'C' key from the zoom window. It allows the user to change the variables on the zoom screen. The variables that can be changed blink when the function prompts for a new value for that variable. The current value can be left unchanged by pressing the return key without entering any number. The function checks for the validity and the range of the input. If the user enters a valid entry, it accepts the new value and modifies the display. It then prompts for a new value for the next variable until all the variables that can be changed are confirmed correct.

4.7.1.3 Freeze

This function is invoked by pressing the 'F' key on

the keyboard. It freezes the simulation and displays the setup times to the left side and machining in all departments. The simulation can be resumed by pressing the 'R' key.

4.7.1.4 Mode

The modes of purchase, setup and move are displayed in the overview window and can be changed by pressing 'ALT_M" at any time. The modes can be toggled by pressing the space bar or can be left as it is by pressing return.

4.7.1.5 Pace Keys

The pace at which the simulation is performed can be changed by the gray '+' and '-' keys on the numeric keypad. There are 5 different paces available; pace 1 being the slowest and pace 5 the fastest.

4.7.1.6 Schedule

This function is used to schedule user specified events. Apart from that it is also used to save the simulation state by choosing the 'Write' option. This is invoked by pressing 'W' key on the keyboard from the schedule menu. The menu also has the possible user schedulable events namely, 'Purchase', 'Setup', 'Load', and 'Move'. A required event can be scheduled by pressing the corresponding highlighted letter. The function then prompts for the event variables to be associated with that

event type. In addition, it prompts for the scheduled time in the form of week, day, hour and minute.

4.7.1.7 Quit

This function is invoked by pressing the 'Q' key when the simulation is in progress or when it is frozen. The program is terminated gracefully after confirmation from the user. The simulation can be resumed normally by answering 'No' for the confirmation.

4.7.2 Calendar Edit Mode

The simulator goes into the edit mode if the 'E' key is pressed when the calendar window is displayed. A sample layout of the calendar window is shown in Figure 4.4. The menu changes, and the user now can choose to 'DeleteEvent', 'AddEvent', or quit edit mode. Pressing the 'ESCape' key quits the edit mode. The functions 'DeleteEvent' and 'AddEvent' are described in the next two sections.

4.7.2.1 Delete Event

The user can delete any mandatory event, from the calendar by selecting the event to be deleted and pressing the 'Del' key in the numeric keypad. The selected event blinks and is marked by a '*' in the modifier field if deleted. The events are selected by the scan keys 'PgUp', 'PgDn', 'up arrow' and 'down arrow'. A deleted event can be undeleted by selecting that event and pressing the 'Ins'

```
$  0:0: 5:11 clock_tick
$  0:0: 5:11 purchase   raw: 5 for ptype E qty: 5
$  0:0: 5:12 unload     mach 3 in dept 1
@  0:0: 5:12 load       mach 3 in dept 1 setup for  D
$  0:0: 5:12 clock_tick
$  0:0: 5:13 clock_tick
$  0:0: 5:14 clock_tick
$  0:0: 5:15 clock_tick
$  0:0: 5:15 unload     mach 1 in dept 3
@  0:0: 5:15 load       mach 1 in dept 3 setup for  B
$  0:0: 5:15 unload     mach 1 in dept 1
@  0:0: 5:15 load       mach 1 in dept 1 setup for  A
$  0:0: 5:20 unload     mach 1 in dept 2
@  0:0: 5:20 load       mach 1 in dept 2 setup for  E
@  0:0: 5:21 load       mach 2 in dept 1 setup for  C
$  0:0: 5:21 unload     mach 3 in dept 3
@  0:0: 5:21 load       mach 3 in dept 3 setup for  A
$  0:0: 5:22 unload     mach 1 in dept 5
@  0:0: 5:22 load       mach 1 in dept 5 setup for  E
@  0:0: 5:26 load       mach 1 in dept 4 setup for  B
@  0:0: 6:34 load       mach 2 in dept 3 setup for  D
@  0:0: B:37 load       mach 2 in dept 4 setup for  E

 F1Overview  F2Zoom  F4Stat  Edit  Resume  Quit  <+/->Pace
```

Figure 4.4 Sample layout of the Calendar Window

key on the numeric keypad. Pressing the 'ESCape' key after deletion is done, quits 'DeleteEvent'.

4.7.2.2 Add Event

This function is used to schedule user specified events just like 'Schedule' menu. It doesn't have the 'Write' option. Refer to explanation in section 4.7.1.6.

4.7.3 Functions Which Don't Affect The Simulation

The following sections will discuss the functions which do not affect the simulation.

4.7.3.1 Functions To Change The Display Window

The desired display window can be selected by pressing the function key associated with it. The simulator uses four windows to output the results of the simulation. They are listed with the associated keys below:

'F1' - Overview Window

'F2' - Zoom Window

'F3' - Calendar Window

'F4' - Statistics Window

where 'F1', 'F2', 'F3' and 'F4' refer to the function keys on the keyboard.

4.7.3.2 Print Function

This function dumps the current screen on the printer. It does the same thing as a "PrtSc" (PrintScreen) key. This function freezes the simulation until the printing is

complete if the simulation is not already frozen.

Certain special characters are displayed in the overview and the zoom windows. To get a proper printout of these two screens, the printer (usually any EPSON Printers) needs to be set to IBM Proprinter mode as follows:

Several tiny switches called DIP switches are inside any EPSON printer. These switches are under the access cover on the right side of the printer. To remove the cover it should be pressed down sideways with the palm of your hand as shown in Figure 4.5. The figure also shows the location of the switches and their factory settings. Always the power should be turned OFF before any changes are made in the setting of any of these switches. Any changes made while the power is on will be ignored until the printer is turned off and back on. So all switches must be set with the power off. The following DIP switches should be reset to select the Proprinter mode:

DIP Switch	Factory Setting	Proprinter Setting
1-4	ON	OFF
2-1	ON	OFF
2-3	OFF	ON
1-6 1-7 1-8	ON ON ON	Set any one of these three switches to OFF.

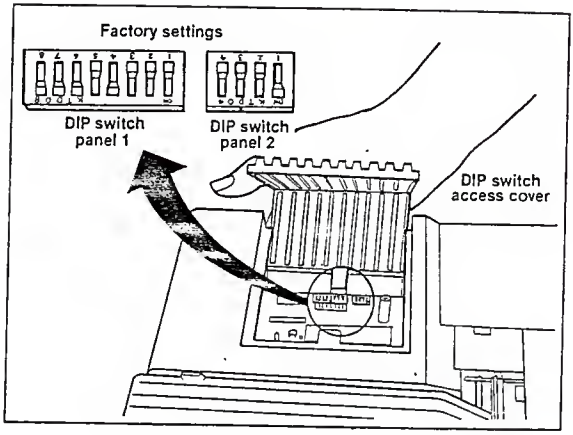


Figure 4.5 DIP Switch Location for FX-286
Epson Printer

4.7.3.3 Scan Keys

The scan keys 'PgUp', 'PgDn', 'up arrow' and 'down arrow' can be operated only in the calendar window. They are used to scan through the calendar list. They can be used when the simulation is in progress or when it is frozen.

4.8 Summary of Menu Functions

A summary of all the menu functions of the simulator is given in the form of a Menu Map in Figure 4.6.

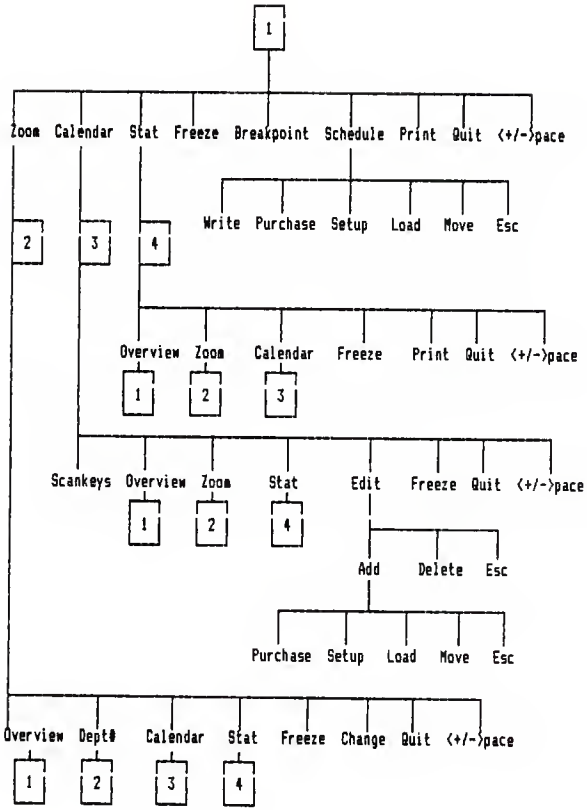


Figure 4.6 Menu Map of the Simulator

CHAPTER 5

SUGGESTIONS FOR EXPANDING THE SIMULATOR

5.1 Introduction

Due to the modular design of the program, further expansion of the simulator to suit various needs is possible without redoing much of the work done already. This Chapter will explore the possible expansions to the simulator and will give some suggestions in implementing them. Some of the aspects of the software engineering will also be discussed here.

5.2 More Departments and More Machines

The overview window in the present simulator does not have enough space to display more than five departments with three machines in each. From the way the department data structure is constructed there is no limit on specifying the number of departments, the simulation runs perfectly normal. But to put things in more perspective for the user, it may be suggested that the overview window itself might have two to three pages. These can be viewed by PgUp or PgDn keys, when the simulation is in progress. With more departments on the overview window, the layout of the statistics window also needs to be adjusted in a similar way for displaying the activation reports.

5.3 Assembly Department And A Manufacturing Cell

An assembly department is a production work-center wherein more than one raw material type is assembled together to give one product. It will have a totally different department structure. The assembly station must have material in all its pre areas before the actual processing starts. This kind of situation will lead to two options: one, where the individual unassembled products could be sold directly and second, where the assembly is done and then sold as the assembled product. Hence a priority can be given in the simulation.

A manufacturing cell can also be built in a similar way. A cell is a workstation wherein a raw material gets processed at different machines, which are all part of the cell, and then it is ready to be sold. The capacity of a manufacturing cell is more than a single workpiece. These two types will really make this simulator a full-fledged interactive production scheduler.

5.4 Overtime For The Shop

Of the variables displayed on the initial display screen, two - "hours per day" and "days per week" - can be changed only before starting the simulation. During the course of the simulation they are fixed. It would be nice to see how the overtime production meets the requirements of master production schedule. In the present simulation,

there is a mandatory breakpoint at the end of each day. At this point, the program could ask whether the user wants the shop to work overtime. If the response is positive, then the program should adjust the timing for all of the future events in the calendar list.

5.5 Pull System of Production Control

If work is completed as scheduled, it is sent from a work center to the next location at which parts are scheduled for use. This is a "push system". Make the parts and send them to where they are next needed, or to inventory, thus pushing material through production according to schedule. This is the strategy scheduled for the development of the present simulator.

In the "Pull System" of production control, the final assembly goes to the preceding process to obtain the necessary parts, at the necessary time, for a specific product assembly. This signals the preceding process to produce (i.e., to replace) the parts withdrawn by the following process. For the production of these parts, the preceding process obtains the necessary parts from the process further preceding it. This implementation can be achieved by overwriting the processor for this simulator. The transfer events are the important events need to be taken care of.

5.6 Creation of Input File

The simulation is started from an input file which has values for all the variables which define the problem. It was mentioned in Chapter 4 that this file should be of a specific format for the program to understand it. It would be desirable if the user is able to create a valid input file through a front-end program. This program for creating the input file can query the user for various inputs and write a valid input file for the simulator just as "WRITE" does so that the user need not check its validity after creation.

5.7 Random Variations in Setup and Machining Times

The simulator discussed here does not have the capability of introducing random variabilities to the setup and machining times. Random variations were avoided to aid in easier reliability checking and faster program development. Addition of this feature involves only a few lines of extra code. A function for generating pseudo random numbers should be written. This function should add variations to the setup and machining times with the user specified times as the mean value.

5.8 Change Setup and Machining Times During Simulation

Allowing changes to the setup and machining times when the simulation is in progress will make it easier to evaluate design alternatives. It can avoid starting up the

simulation again with a new input file with changed setup and machining times. Variations in the raw material costs and product prices during the simulation will also help in evaluating design alternatives.

5.9 Other Features

Other features which will enhance the appearance of the simulator are on-line help and a pull down menu system. Since both of the above enhancements involve similar operations like copying a portion of the screen and restoring it, they can be implemented relatively easily by developing a library function for the purpose.

5.10 Software Engineering

Software engineering is an emerging discipline whose goal is to produce reliable software products in a cost-effective manner. The domain of software engineering includes three intrinsic areas of concern (Goldberg, 1986): software reliability, software management, and programmer productivity. Many practitioners favor software reliability and the drive to produce error-free code as the primary purpose of the software engineering.

One of the major concerns with the current practice of software engineering is an absence of predictability. There is no sound, scientific way of predicting accurately how a software system will behave when it runs.

Within current software engineering practice, the only sound way to make a precise, accurate prediction about a software system is to build it and run it (Good, 1986). There is no way to predict accurately how a system will behave before it can be run. So design flaws often are detected only after a large investment has been made to develop the system to a point where it can be run. A system that can be run can be tested on a set of trial cases. If the system is deterministic, a trial run on a specific test case provides a precise, accurate prediction about how the system will behave in that one case. If the system is re-run on exactly the same case, it will behave in exactly the same way. However, there is no way to predict, from the observed behavior of a finite number of test cases, how the system will behave in any other case. If the system is non-deterministic, the system will not even necessarily repeat its observed behavior on a test case. So in current engineering practice, predicting that a software system will run according to specification is based entirely on subjective, human judgment rather than on objective, scientific fact.

As this simulator was being developed there were quite a few problems which might need a mention here. Specifically, one has to be very careful when dealing with pointers. If a pointer is allocated a certain memory in any routine, one has to make sure that its memory must be

freed at the end of the program by a function called "free". Similarly a temporary pointer must be equated to "NULL" after it has been used and the program is about to be terminated. NULL is a memory address pointing to nothing. Since this program has been written under Microsoft C environment, one could use "codeview" debugger to find any syntax errors. Codeview comes with the Microsoft C compiler.

5.11 Conclusions

Various stages involved in the development of the Interactive Production Scheduler were discussed. The simulator, which is the result of this effort, allows the user to control interactively the flow of the simulation when it is actually in progress.

This simulator will be helpful in training of production management and to facilitate instruction in production scheduling by aiding the visualization of the effects of changes in a production system. It is also a good instructional tool for decision-making. Suggestions for enhancement of the simulator is also presented along with the technical details involved in its operation. These suggestions along with the documented source code of the simulator presented in the Appendix, can help in the future enhancement of the simulator or in development of a related application.

REFERENCES

(1) Buffa, E.S., Modern Production Management, New York: John Wiley, 1961.

(2) Duncan, Ray, Advanced MS DOS Programming, Microsoft Press, Washington, 1986.

(3) Fox, E. Robert, "OPT-An Answer for America, Part II", Inventories and Production Management, Nov-Dec 1982. pp 10-19.

(4) Fox, E. Robert, "OPT-An Answer for America, Part IV, Leapfrogging the Japanese", Inventories and Production Management, March-April 1983. pp 11-23.

(5) Goldberg, R., "Software Engineering: An emerging discipline", IBM Systems Journal, Vol.25, Nos 3/4, 1986.

(6) Goldratt, E.M., "The Unbalanced Plant", American Production and Inventory Control Society Conference Proceedings, 1981. pp 195-199.

(7) Goode, D.I., "Mechanical Proofs about Computer Programs", Readings in Artificial Intelligence and Software Engineering, 1986. pp 65-70.

(8) Hansen, Augie, Proficient C, Microsoft Press, Washington, 1987.

(9) Jones, W. Alfred, "The Design of Interactive simulations", Annual Simulation Symposium, 1985. pp 217-233.

(10) Lulu, J., "Just-in-time Production and Process Reliability", Annual Simulation Symposium, 1984. pp 237-241.

(11) Meleton, P. Marcus, "OPT-Fantasy or Breakthrough?", Production and Inventory Management, Second Quarter, 1986. pp 13-21.

(12) Microsoft C5.1 Optimizing Compiler Run-Time Library Reference, Microsoft Corporation, 1987.

(13) Mize, J.H., PROSIM-V: Administrator's Manual: Production System Simulator. Englewood Cliffs, NJ:Prentice-Hall, 1971.

(14) Partridge, S.E., and D.Sculli, "Designing Management Games For Production Management Training", Simulation and Games, No.15, 1984. pp 328-341.

APPENDIX "A"

PROJECT MAKEFILE

```
CC1= cl /c /AM
CC2= cl
OBJS1= opt+calendar+process+setscr+s_conv+utils+change
OBJS2= input+freeze+stat+plib+print+define+zoom
PATH= d:\include\user\

OPTOBJS1= opt.obj calendar.obj process.obj setscr.obj s_conv.obj utils.obj change.obj
OPTOBJS2= input.obj freeze.obj stat.obj plib.obj print.obj define.obj zoom.obj

opt :
     kmake optobjs1
     kmake optobjs2
     kmake opt.exe

optobjs1 :   $(OPTOBJS1)
optobjs2 :   $(OPTOBJS2)

opt.exe :
     link $(OBJS1)+$(OBJS2),,nul,video;

opt.obj :    opt.c $(PATH)vi.h $(PATH)keydefs.h $(PATH)opt.h $(PATH)std.h
     $(CC1) opt.c
process.obj :        process.c $(PATH)vi.h $(PATH)process.h $(PATH)opt.h $(PATH)std.h
     $(CC1) process.c
calendar.obj :       calendar.c $(PATH)calendar.h $(PATH)opt.h $(PATH)vi.h $(PATH)keydefs.h $(PATH)std.h
     $(CC1) calendar.c
s_conv.obj : s_conv.c $(PATH)calendar.h $(PATH)opt.h
     $(CC1) s_conv.c
utils.obj :  utils.c $(PATH)std.h $(PATH)doslib.h $(PATH)keydefs.h $(PATH)bioslib.h $(PATH)opt.h
             $(PATH)vi.h
     $(CC1) utils.c
setscr.obj : setscr.c $(PATH)opt.h $(PATH)vi.h
     $(CC1) setscr.c
input.obj :  input.c $(PATH)opt.h
     $(CC1) input.c
tstfscan.obj :       tstfscan.c $(PATH)opt.h
     $(CC1) tstfscan.c
freeze.obj : freeze.c $(PATH)opt.h $(PATH)vi.h $(PATH)std.h
     $(CC1) freeze.c
stat.obj :   stat.c $(PATH)opt.h $(PATH)vi.h
     $(CC1) stat.c
plib.obj :   plib.asm
     masm plib.asm,,nul,nul;
print.obj :  print.c $(PATH)bioslib.h $(PATH)std.h $(PATH)vi.h $(PATH)keydefs.h $(PATH)opt.h
     $(CC1) print.c
define.obj : define.c $(PATH)define.h
     $(CC1) define.c
zoom.obj :   zoom.c $(PATH)keydefs.h $(PATH)opt.h $(PATH)vi.h $(PATH)std.h
     $(CC1) zoom.c
change.obj : change.c $(PATH)define.h $(PATH)vi.h $(PATH)opt.h
     $(CC1) change.c
```

PROJECT HEADER FILES

```
/*********************************************************************
 *                                                                   *
 *          Bioslib.h -- Header file for BIOS functions.             *
 *                                                                   *
 *          This file contains the definition for BIOS               *
 *          functions used by the OPT Simulator.                     *
 *                                                                   *
 *          Source : Proficient C                                    *
 *********************************************  **********************/
#define PRINT_SCRN    . 05    /* BIOS interrupts */
#defin_ TOD_INIT     0x0B
#define KEYBD_INIT    0x09
#define DISK_INIT     0x0E
#define VIDEO_IO      0x10
#define EQUIP_CK      0x11
#define MEM_SIZE      0x12
#define DISK_IO       0x13
#define RS232_IO      0x14
#define CASS_IO       0x15
#define KEYBD_IO      0x16
#define PRINT_IO      0x17
#define TOD           0x1A
#define VIDEO_INIT    0x1D
#define GRAPHICS      0x1F

#define SET_MODE      0       /* video routine numbers */
#define CUR_TYPE      1       /* (placed in register AH before a */
#define CUR_POS       2       /* BIOS interrupt 10H) */
#define GET_CUR       3
#define LPEN_POS      4
#define SET_PAGE      5
#define SCROLL_UP     6
#define SCROLL_DN     7
#define READ_CHAR_ATTR        8
#define WRITE_CHAR_ATTR 9
#define WRITE_CHAR    10
#define PALETTE       11
#define WRITE_DOT     12
#define READ_DOT      13
#define WRITE_TTY     14
#define GET_STATE     15
#define ALT_FUNCTION 18       /* EGA only */
#define WRITE_STR     19      /* AT only */

#define RESET_DISK    0       /* disk routine numbers */
#define DISK_STATUS   1
#define READ_SECTOR   2
#define WRITE_SECTOR 3
#define VERIFY_SECTOR         4
#define FORMAT_TRACK 5
```

```
#define KBD_READ     0       /* keyboard routine numbers */
#define KBD_READY    1
#define KBD_STATUS   2
```

```
/*********************************************************************
 *                                                                   *
 *          Calendar.h -- Header file for Calendar.c                 *
 *                                                                   *
 *          This file contains the declaration of the                *
 *          calendar structure.                                      *
 *                                                                   *
 *          Author : M. Arunachalam and M. Theegala                  *
 *********************************************************************/
#ifndef CALENDAR_H
#define CALENDAR_H

/* Constants */
#define N_LINES      21      /* Maximum number of lines in calendar page */

#define CALENDAR struct calendar

CALENDAR{
     char modifier;
     long unsigned time;     /* Time in minutes */
     int weeks;
     int days;
     int hrs;
     int mins;
     char *fnc_ptr;          /* Function pointer */
     int a;
     int b;
     int c;
     int d;
     CALENDAR *prev;
     CALENDAR *next;
};

#endif              /* CALENDAR_H */
```

```
#ifndef DEFINE_H
#define DEFINE_H

/* Declaration of integer write structure */
typedef struct{
     char var[20];
     int *value;
}INT_WRITE;

INT_WRITE int_write[];

/*  Declaration of screen location structure  for  the
variables */
typedef struct{
     char var[20];
     int n_cols;
     int row;
     int col;
}VAR_LOCATION;

VAR_LOCATION var_location[];

#endif       /* DEFINE_H */
```

```
/*********************************************************************
 *                                                                   *
 *          Doslib.h -- Header file for DOS Interrupts.              *
 *                                                                   *
 *          This file contains the definition of DOS                 *
 *          interrupts used by the DPT Simulator.                    *
 *                                                                   *
 *          Source : Proficient C                                    *
 *********************************************************************/
/* DOS interrupts */
#define PGM_TERMINATE       0x20
#define BDOS_REQ    0x21
#define TERMINATE_AOOR      0x22
#define CB_EXIT_AODR 0x23
#define CRITICAL_ERR 0x24
#define ABS_DISK_READ       0x25
#define ABS_DISK_WRITE      0x26
#define PGM_TERM_RES 0x27

/* DOS functions -- usually placed in ah register for CB6 calls */
#define PGM                 0x0
#define KEYIN_ECHO_CB               0x1
#define DSPY_CHAR           0x2
#define AUX_IN                      0x3
#define AUX_OUT                     0x4
#define PRNT_OUT            0x5
#define KEYIN_ECHO          0x6
#define KEYIN                       0x7
#define KEYIN_CB            0xB
#define DSPY_STR            0x9
#define KEYIN_BUF           0xA
#define CH_READY            0xB
#define SET_CH                      0xC
#define DISK_RESET          0x0
#define SELECT_DISK         0xE
#define DPEN_FILE           0xF
#define CLOSE_FILE          0x10
#define FIRST_FM            0x11
#define NEXT_FM                     0x12
#define DELETE_FILE         0x13
#define READ_SEQ            0x14
#define WRITE_SEQ           0x15
#define CREATE_FILE         0x16
#define RENAME_FILE         0x17
            /* 0x18 reserved */
#define CURRENT_DISK        0x19
#define SET_DTA                     0x1A
#define GFLT_FAT_INFO               0x1B
#define FAT_INFO            0x1C
            /* 0x10 - 0x20 reserved */
#define READ_RANDOM         0x21
```

```
#define WRITE_RANDOM          0x22
#define FILE_SIZE             0x23
#define SET_INT_VEC           0x25
#define NEW_PSP_SEG           0x26
#define RAND_BLK_READ                 0x27
#define RAND_BLK_WRITE                0x28
#define PARSE_FILENAME                0x29
#define GET_DATE              0x2A
#define SET_DATE              0x2B
#define GET_TIME              0x2C
#define SET_TIME              0x2D
#define TOGGLE_VERIFY                 0x2E
#define GET_DTA                       0x2F
#define DOS_VERSION           0x30
#define PSP_TERM_KEEP                 0x31
            /* 0x32 reserved */
#define CB_CHECK              0x33
            /* 0x34 reserved */
#define GET_INTR              0x35
#define GET_FREE_SPACE                0x36
            /* 0x37 reserved */
#define INTL_INFO             0x38
#define MKDIR                         0x39
#define RMDIR                         0x3A
#define CHDIR                         0x3B
#define CREAT                         0x3C
#define OPEN_FD                       0x3D
#define CLOSE_FD              0x3E
#define WRITE_FD              0x40
#define UNLINK                        0x41
#define LSEEK                         0x42
#define CHMOD                         0x43
#define IOCTL                         0x44
#define DUP                   0x45
#define FORCE_DUP             0x46
#define GET_CUR_DIR           0x47
#define ALLOC                         0x48
#define FREE                  0x49
#define SET_BLOCK             0x4A
#define EXEC                  0x4B
#define EXIT                  0x4C
#define WAIT                  0x4D
#define FIND_FIRST            0x4E
#define FIND_NEXT             0x4F
            /* 0x50 - 53 reserved */
#define VERIFY_STATE          0x54
            /* 0x55 reserved */
#define RENAME                        0x56
#define FILE_MTIME            0x57
```

```
/*********************************************************************
 *                                                                   *
 *          Keydefs.h -- Header file for keyboard interrupts.        *
 *                                                                   *
 *          This file contains the definition of keyboard            *
 *          interrupts and values for special keys on IBM            *
 *          PC and clones.                                           *
 *                                                                   *
 *          Source : Proficient C                                    *
 *********************************************************************/
#define XF   0x100          /* extended key flag */

#define K_F1 59 | XF
#define K_F2 60 | XF
#define K_F3 61 | XF
#define K_F4 62 | XF
#define K_F5 63 | XF
#define K_F6 64 | XF
#define K_F7 65 | XF
#define K_F8 66 | XF
#define K_F9 67 | XF
#define K_F10       68 | XF

#define K_SF1       84 | XF         /* shifted function keys */
#define K_SF2       85 | XF
#define K_SF3       86 | XF
#define K_SF4       87 | XF
#define K_SF5       88 | XF
#define K_SF6       89 | XF
#define K_SF7       90 | XF
#define K_SF8       91 | XF
#define K_SF9       92 | XF
#define K_SF10      93 | XF

#define K_CF1       94 | XF         /* control function keys */
#define K_CF2       95 | XF
#define K_CF3       96 | XF
#define K_CF4       97 | XF
#define K_CF5       98 | XF
#define K_CF6       99 | XF
#define K_CF7       100 | XF
#define K_CF8       101 | XF
#define K_CF9       102 | XF
#define K_CF10      103 | XF

#define K_AF1       104 | XF        /* alternate function keys */
#define K_AF2       105 | XF
#define K_AF3       106 | XF
#define K_AF4       107 | XF
#define K_AF5       108 | XF
#define K_AF6       109 | XF
```

```
#define K_AF7       110 | XF
#define K_AF8       111 | XF
#define K_AF9       112 | XF
#define K_AF10      113 | XF

#define K_HOME      71 | XF         /* cursor keypad (NumLock off; not shifted) */
#define K_END       79 | XF
#define K_PGUP      73 | XF
#define K_PGDN      81 | XF
#define K_LEFT      75 | XF
#define K_RIGHT     77 | XF
#define K_UP 72 | XF
#define K_DOWN      80 | XF

#define K_CHOME     119 | XF        /* control cursor keypad */
#define K_CEND      117 | XF
#define K_CPGUP     132 | XF
#define K_CPGDN     118 | XF
#define K_CLEFT     115 | XF
#define K_CRGHT     116 | XF

#define K_CTRLA 1           /* standard control keys */
#define K_CTRLB     2
#define K_CTRLC     3
#define K_CTRLD     4
#define K_CTRLE     5
#define K_CTRLF     6
#define K_CTRLG     7
#define K_CTRLH     8
#define K_CTRLI     9
#define K_CTRLJ     10
#define K_CTRLK     11
#define K_CTRLL     12
#define K_CTRLM     13
#define K_CTRLN     14
#define K_CTRLO     15
#define K_CTRLP     16
#define K_CTRLQ     17
#define K_CTRLR     18
#define K_CTRLS     19
#define K_CTRLT     20
#define K_CTRLU     21
#define K_CTRLV     22
#define K_CTRLW     23
#define K_CTRLX     24
#define K_CTRLY     25
#define K_CTRLZ     26

#define K_ALTA      30 | XF         /* alternate keys */
#define K_ALTB      48 | XF
#define K_ALTC      46 | XF
```

```
#define K_ALTD      32 | XF
#define K_ALTE      18 | XF
#define K_ALTF      33 | XF
#define K_ALTG      34 | XF
#define K_ALTH      35 | XF
#define K_ALTI      23 | XF
#define K_ALTJ      36 | XF
#define K_ALTK      37 | XF
#define K_ALTL      38 | XF
#define K_ALTM      50 | XF
#define K_ALTN      49 | XF
#define K_ALTO      24 | XF
#define K_ALTP      25 | XF
#define K_ALTQ      16 | XF
#define K_ALTR      19 | XF
#define K_ALTS      31 | XF
#define K_ALTT      20 | XF
#define K_ALTU      22 | XF
#define K_ALTV      47 | XF
#define K_ALTW      17 | XF
#define K_ALTX      45 | XF
#define K_ALTY      21 | XF
#define K_ALTZ      44 | XF

#define K_ALT1            120 | XF   /* additional alternate key combinations */
#define K_ALT2            120 | XF
#define K_ALT3            120 | XF
#define K_ALT4            120 | XF
#define K_ALT5            120 | XF
#define K_ALT6            120 | XF
#define K_ALT7            120 | XF
#define K_ALT8            120 | XF
#define K_ALT9            120 | XF
#define K_ALT0            120 | XF
#define K_ALTDASH   120 | XF
#define K_ALTEQU    120 | XF

#define K_ESC             27          /* miscellaneous special keys */
#define K_SPACE           32
#define K_INS             82 | XF
#define K_DEL             83 | XF
#define K_TAB             K_CTRLI
#define K_BACKTAB   K_CTRLO

#define K_CTRL_PRTSC 114 | XF       /* printer echoing toggle */

#define K_RETURN    13              /* return key variations */
#define K_SRETURN   13
#define K_CRETURN   10
```

```
/*********************************************************************
 *                                                                   *
 *          Dpt.h -- Header file for the OPT Simulator               *
 *                                                                   *
 *          This file contains the declaration of all                *
 *          global variables and structures used by the              *
 *          OPT Simulator.                                           *
 *                                                                   *
 *          Author : M. Arunachalam and M. Theegala                  *
 *********************************************************************/
#ifndef OPT_H
#define OPT_H

#define MIN_PER_HR   60L
#define HRS_PER_DAY  ((unsigned long)hrs_per_day)
#define DAYS_PER_WEEK        ((unsigned long)days_per_week)
#define MIN_PER_DAY  (HRS_PER_DAY * MIN_PER_HR)
#define HRS_PER_WEEK (HRS_PER_DAY * DAYS_PER_WEEK)
#define MIN_PER_WEEK (HRS_PER_WEEK * MIN_PER_HR)

#define MIN_INV                      min_inv
#define PURCHASE_LOT_SIZE    purchase_lot_size
#define TRANSFER_BATCH               transfer_batch
#define MAT_HANDLING_COST    mat_handling_cost
#define OTHER_OP_EXPENSE     other_op_expense

#define DELETED              '\x2A'
#define CLOCK_MODIFIER       '\x24'
#define UNLOAD_MODIFIER      '\x24'
#define REG_MODIFIER '\x40'
#define USER_SPECIFIED       'U'
#define USER_DELETED '\xF'
#define UN_UTILIZED  224
#define CLOCK_TICK   "clock_tick"
#define PURCHASE     "purchase"
#define SETUP                "setup"
#define LOAD         "load"
#define UNLOAD               "unload"
#define MOVE         "move"

#define SETTING_UP   'S'
#define PROCESSING   '\xAF'
#define IDLE         '+'

/* Formal definition of the functions */
char *my_malloc(int);                /* Error sensitive malloc() */

/* Global variables */
int pageno;
unsigned long int event_time, time_now;
int week, day, hr, min, week_now, day_now, hr_now, min_now;
```

```
char modifier;
char event_type[20];
int event_var1, event_var2, event_var3, event_var4;
int freezed;                /* TRUE when freezed and FALSE when not */
int deleted;                /* TRUE when event deleted FALSE otherwise */
int user_specified;         /* TRUE if user specified setup */
int processing;                     /* TRUE if machine is processing at the time of
                              user specified setup */
int auto_purchase;          /* TRUE if user wants automatic replenishment
                              of raw materials */
int auto_setup;                     /* TRUE if user wants automatic setups */
int auto_move;                      /* TRUE if user wants automatic material
                              transfer */

int state;

long cash, expense;
int n_depts, n_ptypes, max_ops;
int pace;

int route[10][10];
int stime[2][10][10];
int mtime[2][10][10];

int raw_for_ptype[10];
int routing_row, routing_col, raw_mat_row, raw_mat_col, prod_row, prod_col;
int n_raw_mat, n_products, priority_row, priority_col;
int zoom_row, zoom_col;
int zoom_dept;

int hrs_per_day;
int days_per_week;

int min_inv;
int purchase_lot_size;
int transfer_batch;
int mat_handling_cost;
int other_op_expense;
int stat_pgup;

/* Global structures */
#define MACHINE struct machine
#define PART_INFO struct part_information
#define DEPT struct department
#define RAW struct raw_material
#define PRODUCT     struct products
#define STATS struct statistics

/* Private variables to files which include opt.h */
DEPT *dept_h;
RAW *raw_mat;
```

```
PRODUCT *product;
STATS *stats;

/* Structure for machine */
MACHINE{
     int row;
     int col;
     int mach_no;
     int repeat;
     int setup_for;
     char setup_for_in_char;
     int mstate;
     int user_stopped;
     unsigned long int state_change_time;
     unsigned long int time_idle;
     unsigned long int time_setup;
     unsigned long int time_busy;
     MACHINE *next;
};

/* Structure for part information */
PART_INFO{
     int row;
     int col;
     int repeat;
     int ptype;
     char ptype_in_char;
     int pre;
     int post;
     int dept_no;
     PART_INFO *next;
};

/* Structure for department */
DEPT{
     int dept_no;
     int nmachines;
     int nmach_in_use;
     int row;
     int col;
     MACHINE *mach;
     PART_INFO *part_info;
     DEPT *next;
};

/* Structure for raw materials */
RAW{
     int raw_mat_no;
     int cost;
     int row;
     int col;
```

```
    RAW *next;
};

/* Structure for products */
PRODUCT{
    int prod_no;
    int price;
    int qnty_produced;
    int limit;
    int row;
    int col;
    PRODUCT *next;
};

/* Structure for statistics */
STATS{
    int fin_row;
    int fin_col;
    long start_cash;
    long cash_now;
    long sales_revenue;
    int raw_mat_exp;
    int mat_hand_exp;
    int other_op_exp;
    int net_cash_flow;
};

#endif              /* OPT_H */
```

```
/*********************************************************************
 *                                                                   *
 *           Process.h -- Header file for process.c                  *
 *                                                                   *
 *           This file contains the declaration of the               *
 *           event procedures and initialization of the              *
 *           command table for the event processor.                  *
 *                                                                   *
 *           Author : M. Arunachalam and M. Theegala                 *
 *********************************************************************/
#ifndef PROCESS_H
#define PROCESS_H

/* --- Functions in the cal_fncs command table --- */
int advance_clock(int, int, int, int);
int purchase_rawmat(int, int, int, int);
int setup_machine(int, int, int, int);
int load_machine(int, int, int, int);
int unload_machine(int, int, int, int);
int move_part(int, int, int, int);

typedef struct{
     char name[20];          /* Event types - can be 19 characters long */
     int (*fnc)(int, int, int, int); /* Event procedures */
}CAL_FUNCTIONS;

CAL_FUNCTIONS cal_fncs[] = { "clock_tick",   advance_clock,
                             "purchase",     purchase_rawmat,
                             "setup",        setup_machine,
                             "load",         load_machine,
                             "unload",       unload_machine,
                             "move",         move_part,
                             ""};

#endif              /* PROCESS_H */
```

```
/*********************************************************************
 *                                                                   *
 *          Std.h -- Header file for standard declarations           *
 *                                                                   *
 *          This file contains the definition of                     *
 *          standard constants used by the OPT Simulator.            *
 *                                                                   *
 *          Source : Proficient C                                    *
 *********************************************************************/
/* data type aliases */
#define META short
#define UCHAR       unsigned char
#define UINT unsigned int
#define ULONG       unsigned long
#define USHORT      unsigned short

/* Boolean data type */
typedef enum {
    FALSE, TRUE
} BOOLEAN;

/* function return values and program exit codes */
#define OK   0
#define BAD  1
#define SUCCESS 0
#define FAILURE 1
#define FALSE       0
#define TRUE 1

/* infinite loop */
#define FOREVER while (1)

/* masks */
#define HIBYTE      0xFF00
#define LOBYTE      0x00FF
#define ASCII       0x7F
#define HIBIT       0x80

/* lengths */
#define MAXNAME 8
#define MAXEXT      3
#define MAXLINE     256
#define MAXPATH     64

/* special number */
#define BIGGEST     65535
```

```
/*********************************************************************
 *                                                                   *
 *           Vi.h -- Header file for the video routines              *
 *                                                                   *
 *           This file contains declarations and definitions         *
 *           used by direct video routines in the video              *
 *           library "Video.h"                                       *
 *                                                                   *
 *           Authors : Kent Funk                                     * *
 *********************************************************************/
#ifndef VI_H
#define VI_H

#define VSTAT  0x30A
#define VRBIT  8
#define VSYNC  while ((inp(VSTAT) & VRBIT) == VRBIT); \
                      while ((inp(VSTAT) & VRBIT) != VRBIT)
    /* Screen attributes */
#define NORM 7
#define REV  112
#define INT  8
#define BLNK 128

#define F_BLACK     0
#define F_BLUE          1
#define F_GREEN     2
#define F_CYAN          3
#define F_RED           4
#define F_MAGENTA   5
#define F_BROWN     6
#define F_WHITE     7

#define B_BLACK     0
#define B_BLUE          16
#define B_GREEN     32
#define B_CYAN          48
#define B_RED           64
#define B_MAGENTA   80
#define B_BROWN     96
#define B_WHITE     122

/* screen attributes of entities in the simulator */
#define CLOCK_COLOR  B_BLACK+F_BROWN+INT
#define S_TEXT_COLOR B_BLACK+F_GREEN+INT
#define TITLE_COLOR  B_BLACK+F_WHITE+INT
#define ERROR_COLOR  B_BLACK+F_WHITE+INT+BLNK
#define MSG_COLOR    B_BLACK+F_WHITE+INT
#define CASH_COLOR   B_BLUE+F_BROWN+INT
#define EXPENSE_COLOR       B_RED+F_WHITE+INT
#define MENU_KEY_COLOR      B_BLUE+F_CYAN+INT
#define MENU_FNC_COLOR      B_BLUE+F_WHITE+INT
```

```
#define MACHINE_COLOR        B_BLUE+F_WHITE+INT
#define STIME_COLOR  B_GREEN+F_BLACK
#define MTIME_COLOR  B_GREEN+F_BLACK
#define PRE_COLOR    B_CYAN+F_BLUE
#define POST_COLOR   B_CYAN+F_BLUE
#define ROUTE_COLOR  B_BLACK+F_CYAN+INT
#define PROD_NO_COLOR        B_BLUE+F_CYAN+INT
#define PRODUCT_COLOR        REV                    /* B_WHITE+F_BLUE */
#define PRIORITY_COLOR       B_BLACK+F_CYAN+INT
#define CAL_MOD_COLOR        B_GREEN+F_BROWN+INT
#define CAL_TIME_COLOR       B_BLUE+F_WHITE+INT
#define CAL_FNC_COLOR        B_RED+F_WHITE+INT
#define CAL_REST_COLOR       B_BLUE+F_CYAN
#define STAT_FIN_COLOR       B_RED+F_WHITE+INT
#define STAT_SLS_COLOR       B_MAGENTA+F_WHITE+INT
#define STAT_ACT_COLOR       B_BLUE+F_WHITE+INT

    /* Scrolling directions */
#define SCRL_UP              0
#define SCRL_DOWN    1
#define SCRL_LEFT    2
#define SCRL_RIGHT   3

    /* clear screen macro */
#define vicls(vpage) (viblank(vpage, 0, 0, 24, 79))

    /* Formal screen function definitions */
void viblank(int, int, int, int, int);
void viputs(int, int, int, int, char *);
void viputc(int, int, int, int, char);
void viputcc(int, int, int, char);
void vierasecc(int, int, int);
void viscr(int, int, int, int, int, int);
void vibox(int, int, int, int, int, char, int);

void viprintf(int, int, int, int, char *, ); /* variable args */
void vierrmsg(int, char *, );                        /* variable args */

#endif /* VI_H */
```

PROJECT SOURCE FILES

```
#include <stdio.h>
#include <conio.h>
#include "user\calendar.h"
#include "user\opt.h"
#include "user\vi.h"
#include "user\keydefs.h"
#include "user\std.h"

static CALENDAR *cal_q, *cal_h, *cal_t, *cal_user, *cal_temp;
static CALENDAR *first_line, *last_line;
static int final_line;

/* --- Functions for calendar management --- */
void form_calendar(void);
void set_cal_window(void);
void display_calendar(void);
void get_next_event(void);
void disp_last_line(int);
void display_line(int, CALENDAR*);
void schedule(char*, unsigned long int, int, int, int, int);
void change(char*, int, int, int, int);
void del_event_from_cal(void);
void dispose_calendar(void);
void set_cal_pgup(void);
void set_cal_pgdn(void);
void set_cal_1_up(void);
void set_cal_1_down(void);
void edit_calendar(void);
void delete_event(void);
void user_schedule(void);
void update_modifier(CALENDAR*);
int write_cal_list(char*);
int read_cal_list(FILE*);

/* --- Private functions to calendar.c --- */
static void display_details(int, CALENDAR*);

/* Function to form the calendar */
void form_calendar()
{
     unsigned long int clk_time;
     CALENDAR *temp;

     cal_q = cal_h = cal_t = cal_user = cal_temp = NULL;
     cal_q = (CALENDAR*)my_malloc(sizeof(CALENDAR));
     cal_h = cal_t = cal_user = cal_temp = cal_q;
     cal_h->modifier = CLOCK_MODIFIER;
     cal_h->time = 1L;
     time_to_string(cal_h->time, cal_h);
     cal_h->fnc_ptr = CLOCK_TICK;
     cal_h->a = 0;
```

```
    cal_h->b = 0;
    cal_h->c = 0;
    cal_h->d = 0;
    cal_h->prev = NULL;
    temp = cal_h;
    for(clk_time=2L; clk_time < 6L; clk_time++){
            temp->next = (CALENDAR*)my_malloc(sizeof(CALENDAR));
            temp = temp->next;
            temp->prev = cal_t;
            cal_t = temp;
            temp->modifier = CLOCK_MODIFIER;
            temp->time = clk_time;
            time_to_string(temp->time, temp);
            temp->fnc_ptr = CLOCK_TICK;
            temp->a = 0;
            temp->b = 0;
            temp->c = 0;
            temp->d = 0;
    }
    cal_t->next = NULL;
    temp = NULL;
}

/* Function to set the visual calendar limits */
void set_cal_window()
{
    int nlines;

    first_line = cal_h;
    for(nlines=0, last_line=first_line;\
            last_line->next != NULL && nlines < N_LINES;\
                    last_line = last_line->next, nlines++);
}

/* Function to display the calendar in the calendar window */
void display_calendar()
{
    int i;
    CALENDAR *disp_line;

    viblank(2, 0, 0, 21, 79);
    for(i=0, disp_line=first_line; disp_line != last_line->next;\
                                    disp_line = disp_line->next, i++){
            viprintf(2, i, 0, CAL_MOD_COLOR, "%c ", disp_line->modifier);
            viprintf(2, i, 2, CAL_TIME_COLOR, "%2d:%1d:%2d:%2d ",\
                                    disp_line->weeks, disp_line->days,\
                                    disp_line->hrs, disp_line->mins);
            if(disp_line->mins < 10) viprintf(2, i, 10, CAL_TIME_COLOR, "%c", '0');
            viprintf(2, i,13, CAL_FNC_COLOR, "%s", disp_line->fnc_ptr);
            display_details(i, disp_line);
    }
```

```
    final_line = i-1;
}

/* Function to display the details in the calendar line */
static void display_details(scr_line, cal_ptr)
int scr_line;
CALENDAR *cal_ptr;
{
    char pchar;

    if(strcmp(cal_ptr->fnc_ptr, CLOCK_TICK) == 0){
        viprintf(2, scr_line, 23, CAL_FNC_COLOR, "%c", ' ');
    }
    if(strcmp(cal_ptr->fnc_ptr, PURCHASE) == 0){
        viprintf(2, scr_line, 21, CAL_FNC_COLOR, "%s", "    ");
        viprintf(2, scr_line, 24, CAL_REST_COLOR, "%s", "raw: ");
        viprintf(2, scr_line, 29, CAL_REST_COLOR+INT, "%ld", cal_ptr->a);
        viprintf(2, scr_line, 30, CAL_REST_COLOR, "%s", " for ptype ");
        map_part_type(cal_ptr->b, &pchar);
        viprintf(2, scr_line, 41, CAL_REST_COLOR+INT, "%c", pchar);
        viprintf(2, scr_line, 42, CAL_REST_COLOR, "%s", " qty: ");
        viprintf(2, scr_line, 48, CAL_REST_COLOR+INT, "%d", cal_ptr->c);
    }
    if(strcmp(cal_ptr->fnc_ptr, SETUP) == 0){
        viprintf(2, scr_line, 18, CAL_FNC_COLOR, "%s", "      ");
        viprintf(2, scr_line, 24, CAL_REST_COLOR, "%s", "mach ");
        viprintf(2, scr_line, 29, CAL_REST_COLOR+INT, "%ld", cal_ptr->a);
        viprintf(2, scr_line, 30, CAL_REST_COLOR, "%s", " in dept ");
        viprintf(2, scr_line, 39, CAL_REST_COLOR+INT, "%ld", cal_ptr->b);
        viprintf(2, scr_line, 40, CAL_REST_COLOR, "%s", " for ptype ");
        if(cal_ptr->c > 0){
            viprintf(2, scr_line, 51, CAL_REST_COLOR+INT, "%ld", cal_ptr->c);
        }else{
            viprintf(2, scr_line, 51, CAL_REST_COLOR, "%c", ' ');
        }
        map_part_type(cal_ptr->d, &pchar);
        viprintf(2, scr_line, 52, CAL_REST_COLOR+INT, "%c", pchar);
    }
    if(strcmp(cal_ptr->fnc_ptr, LOAD) == 0){
        viprintf(2, scr_line, 17, CAL_FNC_COLOR, "%s", "       ");
        if(cal_ptr->a == 0){
            viprintf(2, scr_line, 24, CAL_REST_COLOR, "%s", "free mach in dept ");
            viprintf(2, scr_line, 42, CAL_REST_COLOR+INT, "%ld", cal_ptr->b);
            viprintf(2, scr_line, 43, CAL_REST_COLOR, "%s", " setup for ");
            if(cal_ptr->c > 0){
                viprintf(2, scr_line, 54, CAL_REST_COLOR+INT, "%ld", cal_ptr->c);
            }else{
                viprintf(2, scr_line, 54, CAL_REST_COLOR, "%c", ' ');
            }
            map_part_type(cal_ptr->d, &pchar);
            viprintf(2, scr_line, 55, CAL_REST_COLOR+INT, "%c", pchar);
```

```
            }else{
                    viprintf(2, scr_line, 24, CAL_REST_COLOR, "%s", "mach ");
                    viprintf(2, scr_line, 29, CAL_REST_COLOR+INT, "%ld", cal_ptr->a);
                    viprintf(2, scr_line, 30, CAL_REST_COLOR, "%s", " in dept ");
                    viprintf(2, scr_line, 39, CAL_REST_COLOR+INT, "%ld", cal_ptr->b);
                    viprintf(2, scr_line, 40, CAL_REST_COLOR, "%s", " setup for ");
                    if(cal_ptr->c > 0){
                            viprintf(2, scr_line, 51, CAL_REST_COLOR+INT, "%ld", cal_ptr->c);
                    }else{
                            viprintf(2, scr_line, 51, CAL_REST_COLOR, "%c", ' ');
                    }
                    map_part_type(cal_ptr->d, &pchar);
                    viprintf(2, scr_line, 52, CAL_REST_COLOR+INT, "%c", pchar);
            }
    }
    if(strcmp(cal_ptr->fnc_ptr, UNLOAD) == 0){
            viprintf(2, scr_line, 19, CAL_FNC_COLOR, "%s", "     ");
            viprintf(2, scr_line, 24, CAL_REST_COLOR, "%s", "mach ");
            viprintf(2, scr_line, 29, CAL_REST_COLOR+INT, "%ld", cal_ptr->a);
            viprintf(2, scr_line, 30, CAL_REST_COLOR, "%s", " in dept ");
            viprintf(2, scr_line, 39, CAL_REST_COLOR+INT, "%ld", cal_ptr->b);
    }
    if(strcmp(cal_ptr->fnc_ptr, MOVE) == 0){
            viprintf(2, scr_line, 17, CAL_FNC_COLOR, "%s", "       ");
            viprintf(2, scr_line, 24, CAL_REST_COLOR, "%s", "part type ");
            if(cal_ptr->a > 0){
                    viprintf(2, scr_line, 34, CAL_REST_COLOR+INT, "%ld", cal_ptr->a);
            }else{
                    viprintf(2, scr_line, 34, CAL_REST_COLOR, "%c", ' ');
            }
            map_part_type(cal_ptr->b, &pchar);
            viprintf(2, scr_line, 35, CAL_REST_COLOR+INT, "%c", pchar);
            viprintf(2, scr_line, 36, CAL_REST_COLOR, "%s", " from dept ");
            viprintf(2, scr_line, 47, CAL_REST_COLOR+INT, "%ld", cal_ptr->c);
            viprintf(2, scr_line, 48, CAL_REST_COLOR, "%s", " in transfer batch size ");
            viprintf(2, scr_line, 72, CAL_REST_COLOR+INT, "%d", cal_ptr->d);
    }
}

/* Function to display the last line in the calendar list */
void disp_last_line(scr_line)
int scr_line;
{
    viprintf(2, scr_line, 0, CAL_MOD_COLOR, "%c ", last_line->modifier);
    viprintf(2, scr_line, 2, CAL_TIME_COLOR, "%2d:%1d:%2d:%2d ",\
                        last_line->weeks, last_line->days,\
                        last_line->hrs, last_line->mins);
    if(last_line->mins < 10) viprintf(2, scr_line, 10, CAL_TIME_COLOR, "%c", '0');
    viprintf(2, scr_line,13, CAL_FNC_COLOR, "%s", last_line->fnc_ptr);
    display_details(scr_line, last_line);
}
```

```
/* Function to display any line in the calendar list at specified screen line */
void display_line(scr_line, cal_p)
int scr_line;
CALENDAR *cal_p;
{
     viprintf(2, scr_line, 0, CAL_MOD_COLOR, "%c ", cal_p->modifier);
     viprintf(2, scr_line, 2, CAL_TIME_COLOR, "%2d:%1d:%2d:%2d ",\
                                   cal_p->weeks, cal_p->days,\
                                   cal_p->hrs, cal_p->mins);
     if(cal_p->mins < 10) viprintf(2, scr_line, 10, CAL_TIME_COLOR, "%c", '0');
     viprintf(2, scr_line,13, CAL_FNC_COLOR, "%s", cal_p->fnc_ptr);
     display_details(scr_line, cal_p);
}

/* Function to get the next event from the top of the calendar list */
void get_next_event()
{
     CALENDAR *event;

     event = cal_h;
     modifier = event->modifier;
     if(modifier == DELETED){
            deleted = TRUE;
     }else{
            deleted = FALSE;
     }
     event_time = event->time;
     week = event->weeks;
     day  = event->days;
     hr   = event->hrs;
     min  = event->mins;
     strcpy(event_type, event->fnc_ptr);
     event_var1 = event->a;
     event_var2 = event->b;
     event_var3 = event->c;
     event_var4 = event->d;
}

/* Function to set the visual calendar limits if PgUp is pressed */
void set_cal_pgup()
{
     int lines_up, nlines;

     viblank(2, 23, 0, 23, 79);
     for(lines_up = 0; first_line->prev != NULL && lines_up < 20;\
                         first_line = first_line->prev, lines_up++);
     if(lines_up == 0){
            viprintf(2, 23, 0, B_BLACK+F_BROWN+INT, "%s", "You are seeing the top of calendar list");
     }else{
            for(nlines = 0, last_line = first_line;\
```

```
                last_line->next != NULL && nlines < N_LINES;\
                        last_line = last_line->next, nlines++);
        display_calendar();
    }
}

/* Function to set the visual calendar limits if PgDn is pressed */
void set_cal_pgdn()
{
    int lines_down, nlines;

    viblank(2, 23, 0, 23, 79);
    for(lines_down = 0; last_line->next != NULL && lines_down < 20;\
                last_line = last_line->next, lines_down++);
    if(lines_down == 0){
        viprintf(2, 23, 0, B_BLACK+F_BROWN+INT, "%s", "You are seeing the last \
                entries of the calendar list");
    }else{
        for(nlines = 0, first_line = last_line;\
                first_line->prev != NULL && nlines < N_LINES;\
                        first_line = first_line->prev, nlines++);
        display_calendar();
    }
}

/* Function to set the visual calendar limits if Up key is pressed */
void set_cal_1_up()
{
    viblank(2, 23, 0, 23, 79);
    if(first_line->prev == NULL){
        viprintf(2, 23, 0, B_BLACK+F_BROWN+INT, "%s", "You are seeing the top \
                line in the calendar list");
    }else{
        first_line = first_line->prev;
        last_line = last_line->prev;
        vicopy(2, SCRL_DOWN, 0, 21);
        display_line(0, first_line);
    }
    cal_user = first_line;
}

/* Function to set the visual calendar limits if Down key is pressed */
void set_cal_1_down()
{
    viblank(2, 23, 0, 23, 79);
    if(last_line->next == NULL){
        viprintf(2, 23, 0, B_BLACK+F_BROWN+INT, "%s", "You are seeing the last line \
                in the calendar list");
    }else{
        last_line = last_line->next;
        first_line = first_line->next;
```

```
            vicopy(2, SCRL_UP, 0, 21);
            display_line(21, last_line);
        }
        cal_user = last_line;
}

/* Function to allocate a new event node and link to the calendar list */
void schedule(ev_type, t, var1, var2, var3, var4)
char *ev_type;
unsigned long t;
int var1, var2, var3, var4;
{
        CALENDAR *temp;
        int i, unknown_event;

        unknown_event = 1;
        if(t >= time_now){
                if(strcmp(ev_type, CLOCK_TICK) == 0){
                        temp = (CALENDAR*)my_malloc(sizeof(CALENDAR));
                        temp->modifier = CLOCK_MODIFIER;
                        temp->fnc_ptr = CLOCK_TICK;
                        unknown_event = 0;
                }
                if(strcmp(ev_type, PURCHASE) == 0){
                        temp = (CALENDAR*)my_malloc(sizeof(CALENDAR));
                        if(user_specified){
                                temp->modifier = USER_SPECIFIED;
                        }else{
                                temp->modifier = CLOCK_MODIFIER;
                        }
                        temp->fnc_ptr = PURCHASE;
                        unknown_event = 0;
                }
                if(strcmp(ev_type, SETUP) == 0){
                        temp = (CALENDAR*)my_malloc(sizeof(CALENDAR));
                        if(user_specified){
                                temp->modifier = USER_SPECIFIED;
                        }else{
                                temp->modifier = REG_MODIFIER;
                        }
                        temp->fnc_ptr = SETUP;
                        unknown_event = 0;
                }
                if(strcmp(ev_type, LOAD) == 0){
                        temp = (CALENDAR*)my_malloc(sizeof(CALENDAR));
                        temp->modifier = REG_MODIFIER;
                        temp->fnc_ptr = LOAD;
                        unknown_event = 0;
                }
                if(strcmp(ev_type, UNLOAD) == 0){
                        temp = (CALENDAR*)my_malloc(sizeof(CALENDAR));
```

```
                temp->modifier = UNLOAD_MODIFIER;
                temp->fnc_ptr = UNLOAD;
                unknown_event = 0;
        }
        if(strcmp(ev_type, MOVE) == 0){
                temp = (CALENDAR*)my_malloc(sizeof(CALENDAR));
                temp->modifier = REG_MODIFIER;
                temp->fnc_ptr = MOVE;
                unknown_event = 0;
        }
        if(unknown_event) return;
        temp->time = t;
        time_to_string(temp->time, temp);
        temp->a = var1;
        temp->b = var2;
        temp->c = var3;
        temp->d = var4;
        for(cal_temp=cal_h;\
                cal_temp->next != NULL && temp->time >= cal_temp->next->time;\
                        cal_temp=cal_temp->next);
        if(cal_temp == cal_h){
                if(cal_h->time > temp->time){
                        cal_temp->prev = temp;
                        temp->next = cal_temp;
                        temp->prev = NULL;
                        cal_q = cal_h = cal_temp = temp;
                }else{
                        temp->next = cal_temp->next;
                        temp->prev = cal_temp;
                        cal_temp->next->prev = temp;
                        cal_temp->next = temp;
                }
        }else{
                if(cal_temp == cal_t){
                        temp->prev = cal_temp;
                        cal_temp->next = temp;
                        temp->next = NULL;
                        cal_t = temp;
                }else{
                        temp->next = cal_temp->next;
                        temp->prev = cal_temp;
                        cal_temp->next->prev = temp;
                        cal_temp->next = temp;
                }
        }
        for(cal_temp = first_line, i=0;\
                (cal_temp != temp) && (cal_temp != last_line);\
                        cal_temp = cal_temp->next, i++);
        if(cal_temp == temp){
/*              if(final_line == N_LINES) viblank(2, final_line, 0, final_line, 79); */
                if(final_line < N_LINES){
```

```
                        final_line++;
                }else{
                        last_line = last_line->prev;
                }
                vicopy(2, SCRL_DOWN, i, final_line);
                display_line(i, temp);
        }
        if((final_line < N_LINES) && (temp == cal_t)){
                final_line++;
                display_line(final_line, temp);
                last_line = last_line->next;
        }
        temp = NULL;
    }
    return;
}

/* Function to schedule a new setup event if the user requests for one */
void change(ev_type, var1, var2, var3, var4)
char *ev_type;
int var1, var2, var3, var4;
{
    CALENDAR *temp;
    DEPT *temp_d;
    MACHINE *temp_m;
    PART_INFO *temp_pi;
    unsigned long int t, timex;
    int waste_setup;
    int prev_ptype, prev_repeat, i;
    int n_pre;
    int ok, def, w, d, h, m;
    char ch;

    restore_cursor();
    vibl ank(pageno,23,0,23,79);
    viprintf(pageno,23,0,B_RED+F_WHITE+INT, "%s", "Give schedule time: \
            (Hit return for automatic)");
    viprintf(pageno,23,47,B_BLACK+F_GREEN+INT, "%s", "week:");
    position_cursor(pageno,23,53);
    ok = FALSE;
    while(!ok){
        ch = getch();
        if(ch == '\r'){
                done_input();
                goto front;
        }
        if(ch < '0' || ch > '4'){
                ok = FALSE;
        }else{
                ok = TRUE;
                fputc(ch,stderr);
```

```
                w = ch - '0';
        }
    }
    viprintf(pageno,23,55,B_BLACK+F_GREEN+INT, "%s", "day:");
    position_cursor(pageno,23,60);
    ok = FALSE;
    while(!ok){
        ch = getch();
        if(ch == '\r'){
            done_input();
            goto front;
        }
        if(ch < '0' || ch > '6'){
            ok = FALSE;
        }else{
            ok = TRUE;
            fputc(ch,stderr);
            d = ch - '0';
        }
    }
    h=0;
    m=0;
    viprintf(pageno,23,62,B_BLACK+F_GREEN+INT, "%s", "hour:");
    h = iprompt(pageno,23,67,2,h,0,23);
    viprintf(pageno,23,70,B_BLACK+F_GREEN+INT, "%s", "min:");
    m = iprompt(pageno,23,74,2,m,0,59);

    string_to_time(&timex,w,d,h,m);
    if(timex <= time_now){
        done_input();
        return;
    }
    def = FALSE;
    goto forward;

front:
    def = TRUE;
forward:
    if(strcmp(ev_type, SETUP) == 0){
        waste_setup = FALSE;
        for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != var2; temp_d = temp_d->next);
        for(temp_m = temp_d->mach; temp_m != NULL && temp_m->mach_no != var1; temp_m = temp_m->next);
        prev_ptype = temp_m->setup_for;
        prev_repeat = temp_m->repeat;
        if(temp_m->mstate == 1){
            waste_setup = TRUE;
        }
        if(waste_setup){
            for(temp = cal_q; (temp != NULL) && !(strcmp(temp->fnc_ptr, LOAD) == 0) &&\
                                (temp->d == prev_ptype) &&\
                                (temp->c == prev_repeat)); temp = temp->next);
```

```
        if(temp != NULL){
                temp->modifier = DELETED;
                if(def){
                        t = time_now+1L;
                }else{
                        t = timex;
                }
        }
}else{
        for(temp = cal_q; (temp != NULL) && !((strcmp(temp->fnc_ptr, LOAD) == 0) &&\
                                (temp->d == prev_ptype) &&\
                                (temp->c == prev_repeat)); temp = temp->next);
        if(temp != NULL){
                temp->modifier = DELETED;
                }
        update_modifier(temp);
        for(temp = cal_q; (temp != NULL) && !((strcmp(temp->fnc_ptr, LOAD) == 0) &&\
                                (temp->a == var1) &&\
                                (temp->b == var2)); temp = temp->next);
        if(temp == NULL){
                for(temp = cal_q; (temp != NULL) &&\
                        !((strcmp(temp->fnc_ptr, SETUP) == 0) && (temp->a == var1) &&\
                                        (temp->b == var2)); temp = temp->next);
                if(temp == NULL){
                        if(!processing){
                                if(def){
                                        t = time_now+1L;
                                }else{
                                        t = timex;
                                }
                        }else{
                                for(temp = cal_q; (temp != NULL) &&\
                                        !((strcmp(temp->fnc_ptr, UNLOAD) == 0) &&\
                                                (temp->a == var1) &&\
                                                (temp->b == var2)); temp = temp->next);
                                if(def){
                                        t = temp->time;
                                }else{
                                        t = timex;
                                }
                        }
                }else{
                        temp->modifier = DELETED;
                        if(def){
                                t = time_now+1L;
                        }else{
                                t = timex;
                        }
                }
        }else{
                temp->modifier = DELETED;
```

```
                        if(def){
                                t = temp->time;
                        }else{
                                t = timex;
                        }
                }
        }
        update_modifier(temp);
        user_specified = TRUE;
        schedule(SETUP, t, var1, var2, var3, var4);
        user_specified = FALSE;
        processing = FALSE;
    }
    if(strcmp(ev_type, LOAD) == 0){
        for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != var2; temp_d = temp_d->next);
        for(temp_pi = temp_d->part_info; temp_pi != NULL && !(temp_pi->ptype == var4 &&\
                                        temp_pi->repeat == var3);\
                                        temp_pi = temp_pi->next);
        if(temp_pi == NULL) print_errorp();
        n_pre = nmach_proc_ptype(temp_d, var3, var4);
        if(def){
                t = time_now + 1L;
        }else{
                t = timex;
        }
        if(temp_pi->pre >= n_pre){
                schedule(LOAD, t, var1, var2, var3, var4);
        }else{
                viprintf(pageno, 23, 25, B_BLACK+F_BROWN+INT, "%s", \
                        "Not enough parts in the pre area");
        }
    }
}

/* Function to update the modifier in the calendar window */
void update_modifier(temp)
CALENDAR *temp;
{
    int i;

    for(i=0, cal_user = first_line;\
        cal_user != temp && cal_user != last_line; cal_user = cal_user->next, i++);
    if(cal_user == temp){
        display_line(i, temp);
    }
}


/* Function to delete the processed event from the calendar */
void del_event_from_cal()
{
```

```
    CALENDAR *event;

    event = cal_h;
    cal_q = event->next;
    cal_h = event->next;
    cal_h->prev = NULL;
    if(cal_temp == event){
            cal_temp = event->next;
    }
    if(cal_user == event){
            cal_user = event->next;
    }
    if(first_line == event){
            first_line = first_line->next;
            vicopy(2, SCRL_UP, 0, final_line);
            viblank(2, final_line, 0, final_line, 79);
            if((final_line == N_LINES)&&(last_line->next != NULL)){
                    last_line = last_line->next;
                    disp_last_line(final_line);
            }else{
                    final_line--;
            }
    }
    free(event);
}

/* Function to dispose calendar before termination of the simulation */
void dispose_calendar()
{
    CALENDAR *temp;

    for(temp = cal_h->next; temp->next != NULL; temp = temp->next){
            free(temp->prev);
    }
    free(temp->prev);
    free(temp);
    temp = cal_h = cal_t = cal_q = cal_user = cal_temp = NULL;
}

/* Function to edit the calendar */
void edit_calendar()
{
    int k;

    draw_menu(7);
    while(1){
            if(kbhit()){
                    k = getkey();
                    switch(k){
                            case 'd':
                            case 'D':
```

```
                        case K_ALTD:
                                delete_event();
                                break;
                        case 'a':
                        case 'A':
                        case K_ALTA:
                                user_schedule();
                                break;
                        case K_ESC:
                                draw_menu(9);
                                return;
                        default:
                                break;
                }
        }
    }
}

/* Interface for user scheduled events */
void user_schedule()
{
    char ch;
    int k;

    draw_menu(10);
    while(1){
        if(kbhit()){
            k = getkey();
            switch(k){
                case K_ESC:
                    if(pageno == 0){
                        done_input();
                        draw_menu(11);
                    }
                    if(pageno == 2){
                        done_input();
                        draw_menu(7);
                    }
                    return;
                    break;
                case 'w':
                case 'W':
                case K_ALTW:
                    write_sim_state();
                    break;
                case 'p':
                case 'P':
                case K_ALTP:
                    user_purchase();
                    break;
                case 's':
```

```
                        case 'S':
                        case K_ALTS:
                                user_setup();
                                break;
                        case 'l':
                        case 'L':
                        case K_ALTL:
                                user_load();
                                break;
                        case 'm':
                        case 'M':
                        case K_ALTM:
                                user_move();
                                break;
                        default:
                                break;
                }
        }
    }
}

/* Function to mark an event as deleted if the user requests */
void delete_event()
{
    int k, scr_line;

    draw_menu(8);
    cal_user = first_line;
    scr_line = 0;
    while(cal_user->modifier == CLOCK_MODIFIER){
        if(scr_line < final_line){
            scr_line++;
            cal_user = cal_user->next;
        }else{
            set_cal_1_down();
        }
    }
    vichgatt(2, scr_line, 0, 80, BLNK);
    while(1){
        if(kbhit()){
            k = getkey();
            switch(k){
                case K_PGUP:
                    viblank(2, 23, 0, 23, 79);
                    vichgatt(2, scr_line, 0, 80, BLNK);
                    set_cal_pgup();
                    scr_line = 0;
                    cal_user = first_line;
                    while((cal_user->modifier == CLOCK_MODIFIER) &&\
                            (cal_user->prev != NULL)){
                        set_cal_1_up();
```

```
        }
        if((cal_user->prev == NULL) &&\
                (cal_user->modifier == CLOCK_MODIFIER)){
                viblank(2, 23, 0, 23, 79);
                viprintf(2, 23, 15, B_BLACK+F_BROWN+INT, "%s",\
                        "No more deletable entries above this");
                while(cal_user->modifier == CLOCK_MODIFIER){
                        if(scr_line < final_line){
                                scr_line++;
                                cal_user = cal_user->next;
                        }else{
                                set_cal_1_down();
                        }
                }
        }
        vichgatt(2, scr_line, 0, 80, BLNK);
        break;
case K_PGDN:
        viblank(2, 23, 0, 23, 79);
        vichgatt(2, scr_line, 0, 80, BLNK);
        set_cal_pgdn();
        scr_line = final_line;
        cal_user = last_line;
        while((cal_user->modifier == CLOCK_MODIFIER) &&\
                   (cal_user->next != NULL)){
                if(scr_line < final_line){
                        scr_line++;
                        cal_user = cal_user->next;
                }else{
                        set_cal_1_down();
                }
        }
        if((cal_user->next == NULL) &&\
                (cal_user->modifier == CLOCK_MODIFIER)){
                viblank(2, 23, 0, 23, 79);
                viprintf(2, 23, 15, B_BLACK+F_BROWN+INT, "%s",\
                        "No more deletable entries below this");
                while(cal_user->modifier == CLOCK_MODIFIER){
                        if(scr_line > 0){
                                scr_line--;
                                cal_user = cal_user->prev;
                        }else{
                                set_cal_1_up();
                        }
                }
        }
        vichgatt(2, scr_line, 0, 80, BLNK);
        break;
case K_UP:
        viblank(2, 23, 0, 23, 79);
        if(scr_line > 0){
```

```
                vichgatt(2, scr_line, D, B0, BLNK);
                scr_line--;
                cal_user = cal_user->prev;
                while((cal_user->modifier == CLOCK_MODIFIER) &&\
                        (cal_user->prev != NULL)){
                        if(scr_line > 0){
                                scr_line--;
                                cal_user = cal_user->prev;
                        }else{
                                set_cal_1_up();
                        }
                }
                if((cal_user->prev == NULL) &&\
                        (cal_user->modifier == CLOCK_MODIFIER)){
                        viblank(2, 23, 0, 23, 79);
                        viprintf(2, 23, 15, B_BLACK+F_BROWN+INT, "%s",\
                                "No more deletable entries above this");
                        while(cal_user->modifier == CLOCK_MODIFIER){
                                if(scr_line < final_line){
                                        scr_line++;
                                        cal_user = cal_user->next;
                                }else{
                                        set_cal_1_down();
                                }
                        }
                }
                vichgatt(2, scr_line, 0, B0, BLNK);
        }else{
                vichgatt(2, scr_line, 0, B0, BLNK);
                set_cal_1_up();
                while((cal_user->modifier == CLOCK_MODIFIER) &&\
                        (cal_user->prev != NULL)){
                        set_cal_1_up();
                }
                if((cal_user->prev == NULL) &&\
                        (cal_user->modifier == CLOCK_MODIFIER)){
                        viblank(2, 23, 0, 23, 79);
                        viprintf(2, 23, 15, B_BLACK+F_BROWN+INT, "%s",\
                                "No more deletable entries above this");
                        while(cal_user->modifier == CLOCK_MODIFIER){
                                if(scr_line < final_line){
                                        scr_line++;
                                        cal_user = cal_user->next;
                                }else{
                                        set_cal_1_down();
                                }
                        }
                }
                vichgatt(2, scr_line, 0, B0, BLNK);
        }
        break;
```

```
case K_DOWN:
        viblank(2, 23, 0, 23, 79);
        if(scr_line < final_line){
                vichgatt(2, scr_line, 0, B0, BLNK);
                scr_line++;
                cal_user = cal_user->next;
                while((cal_user->modifier == CLOCK_MODIFIER) &&\
                        (cal_user->next != NULL)){
                        if(scr_line < final_line){
                                scr_line++;
                                cal_user = cal_user->next;
                        }else{
                                set_cal_1_down();
                        }
                }
                if((cal_user->next == NULL) &&\
                        (cal_user->modifier == CLOCK_MODIFIER)){
                        viblank(2, 23, 0, 23, 79);
                        viprintf(2, 23, 15, B_BLACK+F_BROWN+INT, "%s",\
                                "No more deletable entries below this");
                        while(cal_user->modifier == CLOCK_MODIFIER){
                                if(scr_line > 0){
                                        scr_line--;
                                        cal_user = cal_user->prev;
                                }else{
                                        set_cal_1_up();
                                }
                        }
                }
                vichgatt(2, scr_line, 0, B0, BLNK);
        }else{
                vichgatt(2, scr_line, 0, B0, BLNK);
                set_cal_1_down();
                while((cal_user->modifier == CLOCK_MODIFIER) &&\
                                (cal_user->next != NULL)){
                                set_cal_1_down();
                }
                if((cal_user->next == NULL) &&\
                        (cal_user->modifier == CLOCK_MODIFIER)){
                        viblank(2, 23, 0, 23, 79);
                        viprintf(2, 23, 15, B_BLACK+F_BROWN+INT, "%s",\
                                "No more deletable entries below this");
                        while(cal_user->modifier == CLOCK_MODIFIER){
                                if(scr_line > 0){
                                        scr_line--;
                                        cal_user = cal_user->prev;
                                }else{
                                        set_cal_1_up();
                                }
                        }
                }
```

```
                                        vichgatt(2, scr_line, 0, 80, BLNK);
                                }
                                break;
                        case K_ESC:
                                viblank(2, 23, 0, 23, 79);
                                vichgatt(2, scr_line, 0, 80, BLNK);
                                draw_menu(7);
                                return;
                        case K_DEL:
                                cal_user->modifier = DELETED;
                                display_line(scr_line, cal_user);
                                vichgatt(2, scr_line, 0, 80, BLNK);
                                break;
                        case K_INS:
                                if(strcmp(cal_user->fnc_ptr, SETUP) == 0){
                                        viblank(2, 23, 0, 23, 79);
                                        viprintf(2, 23, 0, MSG_COLOR, "%s", \
                                             "Insert setup through AddEvent");
                                }else{
                                        cal_user->modifier = REG_MODIFIER;
                                        display_line(scr_line, cal_user);
                                        vichgatt(2, scr_line, 0, 80, BLNK);
                                }
                                break;
                        default:
                                break;
                        }
                }
        }
    viblank(2, 23, 0, 23,79);
}

/* Function to write the simulation state of the calendar list */
write_cal_list(filename)
char filename[15];
{
    FILE *fp;
    char s = ' ';

    if((fp = fopen(filename, "at")) == NULL){
            viprintf(pageno,23,55,B_RED+F_WHITE+INT, "%s", "bad file name");
            hide_cursor();
            return(0);
    }

    fprintf(fp, "\n \n");
    fprintf(fp,"%s \n \n","CALENDAR_LIST:");
    fprintf(fp,"%s \n","modifier:     time:week:day:hrs:mins:event_function_ptr:a___b___c___d");
    for(cal_temp = cal_h; cal_temp != NULL; cal_temp = cal_temp->next){
            if(strcmp(cal_temp->fnc_ptr,"clock_tick")==0){
            fprintf(fp, "%c %c %c %c %91u %3d %4d %3d %3d %c %s %c %c %c %3d %3d %3d %3d \n",\
```

```
                s,s,s,cal_temp->modifier,cal_temp->time,cal_temp->weeks,cal_temp->days,\
                cal_temp->hrs,cal_temp->mins,s,cal_temp->fnc_ptr,s,s,s,s,cal_temp->a,\
                cal_temp->b,cal_temp->c,cal_temp->d);
        }
        if(strcmp(cal_temp->fnc_ptr,"purchase")==0){
        fprintf(fp, "%c%c%c%c %9lu%3d%4d%3d%3d %c%s%c%c%c%c %3d%3d%3d%3d\n",\
                s,s,s,s,cal_temp->modifier,cal_temp->time,cal_temp->weeks,cal_temp->days,\
                cal_temp->hrs,cal_temp->mins,s,cal_temp->fnc_ptr,s,s,s,s,s,cal_temp->a,\
                cal_temp->b,cal_temp->c,cal_temp->d);
        }
        if(strcmp(cal_temp->fnc_ptr,"setup")==0){
        fprintf(fp, "%c%c%c%c %9lu%3d%4d%3d%3d %c%s%c%c%c%c%c%c %2d%3d%3d%3d\n",\
                s,s,s,s,cal_temp->modifier,cal_temp->time,cal_temp->weeks,cal_temp->days,\
                cal_temp->hrs,cal_temp->mins,s,cal_temp->fnc_ptr,s,s,s,s,s,s,s,cal_temp->a,\
                cal_temp->b,cal_temp->c,cal_temp->d);
        }
        if(strcmp(cal_temp->fnc_ptr,"load")==0){
        fprintf(fp, "%c%c%c%c %9lu%3d%4d%3d%3d %c%s%c%c%c%c%c%c %3d%3d%3d%3d\n",\
                s,s,s,s,cal_temp->modifier,cal_temp->time,cal_temp->weeks,cal_temp->days,\
                cal_temp->hrs,cal_temp->mins,s,cal_temp->fnc_ptr,s,s,s,s,s,s,s,cal_temp->a,\
                cal_temp->b,cal_temp->c,cal_temp->d);
        }
        if(strcmp(cal_temp->fnc_ptr,"unload")==0){
        fprintf(fp, "%c%c%c%c %9lu%3d%4d%3d%3d %c%s%c%c%c%c%c %3d%3d%3d%3d\n",\
                s,s,s,s,cal_temp->modifier,cal_temp->time,cal_temp->weeks,cal_temp->days,\
                cal_temp->hrs,cal_temp->mins,s,cal_temp->fnc_ptr,s,s,s,s,s,s,cal_temp->a,\
                cal_temp->b,cal_temp->c,cal_temp->d);
        }
        if(strcmp(cal_temp->fnc_ptr,"move")==0){
        fprintf(fp, "%c%c%c%c %9lu%3d%4d%3d%3d %c%s%c%c%c%c%c%c %3d%3d%3d%3d\n",\
                s,s,s,s,cal_temp->modifier,cal_temp->time,cal_temp->weeks,cal_temp->days,\
                cal_temp->hrs,cal_temp->mins,s,cal_temp->fnc_ptr,s,s,s,s,s,s,s,cal_temp->a,\
                cal_temp->b,cal_temp->c,cal_temp->d);
        }
    }
    fclose(fp);
    return(1);
}

/* Function to read the new simulation state of the calendar list */
read_cal_list(fp)
FILE *fp;
{
    CALENDAR *temp;
    int weeks, days, hrs, mins;
    long unsigned time;
    int a, b, c, d, i;
    char fnc_ptr[15], *modifier;

    fscanf(fp, "%*s");
    fscanf(fp, "%*s");
```

```
fscanf(fp, "%*s");
cal_q = cal_h = cal_t = cal_user = cal_temp = NULL;
cal_q = (CALENDAR*)my_malloc(sizeof(CALENDAR));
cal_h = cal_t = cal_user = cal_temp = cal_q;
fscanf(fp, "%ls", modifier);
fscanf(fp, "%ld", &time);
fscanf(fp, "%d", &weeks);
fscanf(fp, "%d", &days);
fscanf(fp, "%d", &hrs);
fscanf(fp, "%d", &mins);
fscanf(fp, "%s", fnc_ptr);
fscanf(fp, "%d", &a);
fscanf(fp, "%d", &b);
fscanf(fp, "%d", &c);
fscanf(fp, "%d", &d);
if(strcmp(fnc_ptr,"clock_tick")==0){
        cal_h->fnc_ptr = CLOCK_TICK;
}
if(strcmp(fnc_ptr,"purchase")==0){
        cal_h->fnc_ptr = PURCHASE;
}
if(strcmp(fnc_ptr,"setup")==0){
        cal_h->fnc_ptr = SETUP;
}
if(strcmp(fnc_ptr,"load")==0){
        cal_h->fnc_ptr = LOAD;
}
if(strcmp(fnc_ptr,"unload")==0){
        cal_h->fnc_ptr = UNLOAD;
}
if(strcmp(fnc_ptr,"move")==0){
        cal_h->fnc_ptr = MOVE;
}

cal_h->modifier = modifier[0];
cal_h->time = time;
cal_h->weeks = weeks;
cal_h->days = days;
cal_h->hrs = hrs;
cal_h->mins = mins;
cal_h->a = a;
cal_h->b = b;
cal_h->c = c;
cal_h->d = d;
cal_h->prev = NULL;
temp = cal_h;
for(i=0; fscanf(fp, "%ls", modifier) != EOF; i++){
        temp->next = (CALENDAR*)my_malloc(sizeof(CALENDAR));
        temp = temp->next;
        temp->prev = cal_t;
        cal_t = temp;
```

```
            temp->modifier = modifier[0];
            fscanf(fp, "%ld %d %d %d %d", &time, &weeks, &days, &hrs, &mins);
            fscanf(fp, "%s", fnc_ptr);
            fscanf(fp, "%d %d %d %d", &a, &b, &c, &d);
            if(strcmp(fnc_ptr,"clock_tick")==0){
                    temp->fnc_ptr = CLOCK_TICK;
            }
            if(strcmp(fnc_ptr,"purchase")==0){
                    temp->fnc_ptr = PURCHASE;
            }
            if(strcmp(fnc_ptr,"setup")==0){
                    temp->fnc_ptr = SETUP;
            }
            if(strcmp(fnc_ptr,"load")==0){
                    temp->fnc_ptr = LOAD;
            }
            if(strcmp(fnc_ptr,"unload")==0){
                    temp->fnc_ptr = UNLOAD;
            }
            if(strcmp(fnc_ptr,"move")==0){
                    temp->fnc_ptr = MOVE;
            }

            temp->time = time;
            temp->weeks = weeks;
            temp->days = days;
            temp->hrs = hrs;
            temp->mins = mins;
            temp->a = a;
            temp->b = b;
            temp->c = c;
            temp->d = d;
    }
    cal_t->next = NULL;
    temp = NULL;
    return(1);
}
```

```
#include <stdio.h>
#include "user\opt.h"
#include "user\define.h"
#include "user\vi.h"

int change_vars(void);

/* Function to change the variables on the zoom window */
int change_vars()
{
     INT_WRITE *ptr;
     VAR_LOCATION *loc;
     int min, max;

     loc = var_location;
     ptr = int_write;
     if(time_now != 0L){
            loc++; ptr++;
            loc++; ptr++;
     }
     while(*loc->var){
            viblank(1, 23, 0, 23, 79);
            viprintf(1, 23, 0, MSG_COLOR, "%s", "Current value of:");
            viprintf(1, 23, 18, MSG_COLOR, "%s", loc->var);
            viprintf(1, 23, 35, B_RED+F_WHITE+INT, "%4d", *ptr->value);
            viprintf(1, 23, 42, MSG_COLOR, "%s", "Enter new value:");
            vichgatt(1, loc->row, loc->col, loc->n_cols, BLNK);
            min = 0;
            max = 999;
            if((strcmp(loc->var, "days_per_week")) == 0){
                   min = 1;
                   max = 7;
            }
            if((strcmp(loc->var, "hrs_per_day")) == 0){
                   min = 1;
                   max = 24;
            }
            if((strcmp(loc->var, "other_op_expense")) == 0){
                   min = 0;
                   max = 9999;
            }
            *ptr->value = ipromptf(1, 23, 59, loc->n_cols, *ptr->value, min, max);
            vichgatt(1, loc->row, loc->col, loc->n_cols, BLNK);
            if((strcmp(loc->var, "days_per_week")) == 0){
                   viprintf(1, loc->row, loc->col, B_RED+F_WHITE+INT, "%1d", *ptr->value);
            }else{
                   if(((strcmp(loc->var, "hrs_per_day")) == 0) ||\
                                 ((strcmp(loc->var, "min_inv") == 0))){
                     viprintf(1,loc->row,loc->col,B_RED+F_WHITE+INT,"%2d",*ptr->value);
                   }else{
                          if((strcmp(loc->var, "other_op_expense")) == 0){
```

```
                                viprintf(1, loc->row, loc->col, B_RED+F_WHITE+INT, \
                                        "%4d", *ptr->value);
                        }else{
                                viprintf(1, loc->row, loc->col, B_RED+F_WHITE+INT, \
                                        "%3d", *ptr->value);
                        }
                }
        }
        loc++;
        ptr++;
    }
    position_cursor(1, 23, 0);
    hide_cursor();
    viblank(1, 23, 0, 23, 79);
}
```

```
#include "user\define.h"

/* Initialize working hours and days */
hrs_per_day = 24;
days_per_week = 7;

/* Initialize other variables */
min_inv = 2;
purchase_lot_size = 5;
transfer_batch = 1;
mat_handling_cost = 1;              /* $1 per transfer */
other_op_expense = 200;             /* $200 per day */

/* Initialize the integer write structure */
INT_WRITE int_write[] = {   "hrs_per_day",          &hrs_per_day,
                            "days_per_week",        &days_per_week,
                            "min_inv",              &min_inv,
                            "purchase_lot_size",    &purchase_lot_size,
                            "transfer_batch",       &transfer_batch,
                            "mat_handling_cost",    &mat_handling_cost,
                            "other_op_expense",     &other_op_expense,
                            "",                     };

/* Initialize the screen location structure */
VAR_LOCATION var_location[] = {     "hrs_per_day",          2,  8, 17,
                            "days_per_week",        1, 10, 18,
                            "min_inv",              2,  3, 77,
                            "purchase_lot_size",    3,  5, 76,
                            "transfer_batch",       3,  7, 76,
                            "mat_handling_cost",    3,  9, 76,
                            "other_op_expense",     4, 11, 75,
                            "",                     };
```

```
#include <stdio.h>
#include "user\opt.h"
#include "user\vi.h"
#include "user\std.h"

/* Function to freeze the simulation */
int freeze()
{
     int row, col;
     DEPT *temp_d;
     MACHINE *temp_m;
     PART_INFO *temp_pi;

     freezed = TRUE;
     draw_menu(5);
     for(temp_d = dept_h; temp_d != NULL; temp_d = temp_d->next){
            for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
                  if((temp_m->mstate == 1)||(temp_m->mstate == 2)){
                        vichgatt(0, temp_m->row+1, temp_m->col+1, 1, BLNK);
                  }
            }
            for(temp_pi = temp_d->part_info; temp_pi != NULL; temp_pi = temp_pi->next){
                  row = temp_pi->row;
                  col = temp_pi->col;
                  viprintf(0, row, col+4, STIME_COLOR, "%3d",\
                        stime[temp_pi->repeat][temp_pi->dept_no-1][temp_pi->ptype-1]);
                  viprintf(0, row, col+12, MTIME_COLOR, "%3d",\
                        mtime[temp_pi->repeat][temp_pi->dept_no-1][temp_pi->ptype-1]);
            }
     }
     stop_zoom_dept();
}

/* Function to resume the simulation */
int unfreeze()
{
     int row, col;
     DEPT *temp_d;
     MACHINE *temp_m;
     PART_INFO *temp_pi;

     freezed = FALSE;
     draw_menu(6);
     for(temp_d = dept_h; temp_d != NULL; temp_d = temp_d->next){
            for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
                  if((temp_m->mstate == 1)||(temp_m->mstate == 2)){
                        vichgatt(0, temp_m->row+1, temp_m->col+1, 1, BLNK);
                  }
            }
            for(temp_pi = temp_d->part_info; temp_pi != NULL; temp_pi = temp_pi->next){
                  row = temp_pi->row;
```

```
                col = temp_pi->col;
                viprintf(0, row, col+4, STIME_COLOR, "%s", "   ");
                viprintf(0, row, col+12, MTIME_COLOR, "%s", "   ");
            }
        }
        start_zoom_dept();
}

/* Function to stop the simulation */
int stop_sim()
{
        DEPT *temp_d;
        MACHINE *temp_m;

        for(temp_d = dept_h; temp_d != NULL; temp_d = temp_d->next){
            for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
                if((temp_m->mstate == 1)||(temp_m->mstate == 2)){
                    if((vichkatt(0, temp_m->row+1, temp_m->col+1) & BLNK) == BLNK){
                        vichgatt(0, temp_m->row+1, temp_m->col+1, 1, BLNK);
                    }
                }
            }
        }
        stop_zoom_dept();
}

/* Function to start the simulation */
int start_sim()
{
        DEPT *temp_d;
        MACHINE *temp_m;

        for(temp_d = dept_h; temp_d != NULL; temp_d = temp_d->next){
            for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
                if((temp_m->mstate == 1)||(temp_m->mstate == 2)){
                    if((vichkatt(0, temp_m->row+1, temp_m->col+1) & BLNK) != BLNK){
                        vichgatt(0, temp_m->row+1, temp_m->col+1, 1, BLNK);
                    }
                }
            }
        }
        start_zoom_dept();
}
```

```
#include <stdio.h>
#include "user\opt.h"

DEPT *temp_d, *temp_last_d;
MACHINE *temp_m;
PART_INFO *temp_pi, *temp_last_pi;
RAW *temp_raw;
PRODUCT *temp_prod;

/* -- functions -- */
int read_data(char*);
int read_sim_state(char*);

/* Function to read input data from the file specified by the user */
int read_data(filename)
char *filename;
{
    FILE *fp;
    int result, ext = 0;
    int i,j,k;
    char pchar;
    int dept_no, nmachines, nmach_in_use, row, col;
    int repeat;
    int raw_mat_no, cost, prod_no, price, limit, ptype, p_index;
    int temp_cash, temp_expense;

    for(i=0; filename[i] != '\0'; i++){
        if(filename[i] == '.') ext = 1;
    }
    if (!ext){
        strcat(filename, ".sim");
    }
    if((fp = fopen(filename, "r")) == NULL){
        printf("%s : bad file name\n", filename);
        exit(0);
    }
    fscanf(fp, "%*s %d", &temp_cash);
    cash = (long)temp_cash;
    fscanf(fp, "%*s %d", &temp_expense);
    expense = (long)temp_expense;
    fscanf(fp, "%*s %d %*s %d", &n_depts, &n_ptypes);
    fscanf(fp, "%*s %d", &max_ops);
    fscanf(fp, "%*s");
    for(i=0; i < n_ptypes; i++){
        for(j=0; j < max_ops+1; j++){
            fscanf(fp, "%d", &route[i][j]);
        }
    }
    fscanf(fp, "%*s %d %*s %d", &routing_row, &routing_col);
    dept_h = (DEPT*)my_malloc(sizeof(DEPT));
    temp_d = dept_h;
```

```
for(i=0; i < n_depts; i++){
        fscanf(fp, "%*s %d", &dept_no);
        fscanf(fp, "%*s %d %*s %d", &nmachines, &nmach_in_use);
        fscanf(fp, "%*s %d %*s %d", &row, &col);
        temp_d->dept_no = dept_no;
        temp_d->nmachines = nmachines;
        temp_d->nmach_in_use = nmach_in_use;
        temp_d->row = row;
        temp_d->col = col;
        temp_d->mach = (MACHINE*)my_malloc(sizeof(MACHINE));
        temp_m = temp_d->mach;
        temp_m->mach_no = 1;
        temp_m->row = row;
        temp_m->col = col;
        temp_m->repeat = 0;
        temp_m->setup_for = UN_UTILIZED;
        map_part_type(temp_m->setup_for, &pchar);
        temp_m->setup_for_in_char = pchar;
        temp_m->mstate = 0;
        temp_m->user_stopped = 0;
        temp_m->state_change_time = 0L;
        temp_m->time_idle = 0L;
        temp_m->time_setup = 0L;
        temp_m->time_busy = 0L;
        for(j=1; j < nmachines; j++){
                temp_m->next = (MACHINE*)my_malloc(sizeof(MACHINE));
                temp_m = temp_m->next;
                temp_m->mach_no = j+1;
                row += 2;
                temp_m->row = row;
                temp_m->col = col;
                temp_m->repeat = 0;
                temp_m->setup_for = UN_UTILIZED;
                map_part_type(temp_m->setup_for, &pchar);
                temp_m->setup_for_in_char = pchar;
                temp_m->mstate = 0;
                temp_m->user_stopped = 0;
                temp_m->state_change_time = 0L;
                temp_m->time_idle = 0L;
                temp_m->time_setup = 0L;
                temp_m->time_busy = 0L;
        }
        temp_m->next = NULL;
        temp_m = NULL;
        temp_d->part_info = (PART_INFO*)my_malloc(sizeof(PART_INFO));
        temp_pi = temp_d->part_info;
        row = temp_pi->row = temp_d->row+1;
        col = temp_pi->col = temp_d->col-9;
        temp_pi->dept_no = dept_no;
        temp_pi->pre = 0;
        temp_pi->post = 0;
```

```
        for(j=0; j < n_ptypes; j++){
                repeat = 0;
                for(k=0; k < max_ops; k++){
                        if(route[j][k] == dept_no){
                                temp_pi->ptype = j+1;
                                temp_pi->repeat = repeat;
                                map_part_type(temp_pi->ptype, &pchar);
                                temp_pi->ptype_in_char = pchar;
                                temp_last_pi = temp_pi;
                                temp_pi->next = (PART_INFO*)my_malloc(sizeof(PART_INFO));
                                temp_pi = temp_pi->next;
                                row++;
                                temp_pi->row = row;
                                temp_pi->col = col;
                                temp_pi->dept_no = dept_no;
                                temp_pi->pre = 0;
                                temp_pi->post = 0;
                                repeat++;
                        }
                }
        }
        free(temp_pi);
        temp_last_pi->next = NULL;
        temp_last_pi = NULL;
        temp_last_d = temp_d;
        temp_d->next = (DEPT*)my_malloc(sizeof(DEPT));
        temp_d = temp_d->next;
}
free(temp_d);
temp_last_d->next = NULL;
temp_last_d = NULL;
fscanf(fp, "%*s");
fscanf(fp, "%*s %d %*s %d", &raw_mat_row, &raw_mat_col);
fscanf(fp, "%*s %d", &n_raw_mat);
fscanf(fp, "%*s");
raw_mat = (RAW*)my_malloc(sizeof(RAW));
temp_raw = raw_mat;
fscanf(fp, "%d %d", &raw_mat_no, &cost);
temp_raw->raw_mat_no = raw_mat_no;
temp_raw->cost = cost;
row = temp_raw->row = raw_mat_row+1;
col = temp_raw->col = raw_mat_col;
for(i=1; i < n_raw_mat; i++){
        temp_raw->next = (RAW*)my_malloc(sizeof(RAW));
        temp_raw = temp_raw->next;
        fscanf(fp, "%d %d", &raw_mat_no, &cost);
        temp_raw->raw_mat_no = raw_mat_no;
        temp_raw->cost = cost;
        temp_raw->row = ++row;
        temp_raw->col = col;
        if((i+1)%3 == 0){
```

```
                row = raw_mat_row;
                col = raw_mat_col+10;
        }
}
temp_raw->next = NULL;
temp_raw = NULL;

fscanf(fp, "%*s");
fscanf(fp, "%*s %d %*s %d", &prod_row, &prod_col);
fscanf(fp, "%*s %d", &n_products);
fscanf(fp, "%*s");
product = (PRODUCT*)my_malloc(sizeof(PRODUCT));
temp_prod = product;
fscanf(fp, "%d %d %d", &prod_no, &price, &limit);
temp_prod->prod_no = prod_no;
temp_prod->price = price;
temp_prod->qnty_produced = 0;
temp_prod->limit = limit;
row = temp_prod->row = prod_row+1;
col = temp_prod->col = prod_col;
for(i=1; i < n_products; i++){
        temp_prod->next = (PRODUCT*)my_malloc(sizeof(PRODUCT));
        temp_prod = temp_prod->next;
        fscanf(fp, "%d %d %d", &prod_no, &price, &limit);
        temp_prod->prod_no = prod_no;
        temp_prod->price = price;
        temp_prod->qnty_produced = 0;
        temp_prod->limit = limit;
        temp_prod->row = ++row;
        temp_prod->col = col;
        if((i+1)%3 == 0){
                row = prod_row;
                col = prod_col+10;
        }
}
temp_prod->next = NULL;
temp_prod = NULL;

fscanf(fp, "%*s");
for(i=0; i < 2; i++){
        fscanf(fp, "%*s");
        for(j=0; j < n_depts; j++){
                for(k=0; k < n_ptypes; k++){
                        fscanf(fp, "%d", &stime[i][j][k]);
                }
        }
}
fscanf(fp, "%*s");
for(i=0; i < 2; i++){
        fscanf(fp, "%*s");
        for(j=0; j < n_depts; j++){
```

```
                for(k=0; k < n_ptypes; k++){
                        fscanf(fp, "%d", &mtime[i][j][k]);
                }
        }
    }
    fscanf(fp, "%*s %*s %*s");
    for(i=0; i < n_ptypes; i++){
        fscanf(fp, "%d", &ptype);
        fscanf(fp, "%d", &raw_for_ptype[ptype-1]);
    }
    fclose(fp);
    return(0);
}

/*********************************************************************/
/* Function to read previously saved simulation state
   data from the file specified by the user */

int read_sim_state(filename)
char *filename;
{
    FILE *fp;
    int result, ext = 0;
    int i, j, k, count = 0;
    char pchar, test[25];
    int dept_no_t, nmachines_t, nmach_in_use_t;
    int raw_mat_no, cost, prod_no, price, limit, ptype_t, p_index;
    int temp_cash, temp_expense, qnty_produced, temp_time_now;
    int row_t, col_t, mach_no_t, repeat_t, setup_for_t;
    int mstate_t, user_stopped_t, state_change_time_t;
    int idle_t, setup_t, busy_t, pre_t, post_t;
    int starting_cash, now_cash, revenue, temp_event_time;
    char *modify;

    for(i=0; filename[i] != '\0'; i++){
        if(filename[i] == '.') ext = 1;
    }
    if (!ext){
        strcat(filename, ".out");
    }
    if((fp = fopen(filename, "r")) == NULL){
        printf("%s : bad file name\n", filename);
        exit(0);
    }
    fscanf(fp, "%*s %d", &temp_cash);
    cash = (long)temp_cash;
    fscanf(fp, "%*s %d", &temp_expense);
    expense = (long)temp_expense;
    fscanf(fp, "%*s %d %*s %d", &n_depts, &n_ptypes);
    fscanf(fp, "%*s %d", &max_ops);
    fscanf(fp, "%*s %d", &pace);
```

```
fscanf(fp, "%*s");
for(i=0; i < n_ptypes; i++){
        for(j=0; j < max_ops+1; j++){
                fscanf(fp, "%d", &route[i][j]);
                if(route[i][j] == n_depts){
                        count++;
                }
        }
}

fscanf(fp, "%*s %d %*s %d", &routing_row, &routing_col);
fscanf(fp, "%*s %d %*s %d", &raw_mat_row, &raw_mat_col);
fscanf(fp, "%*s %d %*s %d", &prod_row, &prod_col);
fscanf(fp, "%*s %d %*s %d", &n_raw_mat, &n_products);
fscanf(fp, "%*s %d %*s %d", &priority_row, &priority_col);


dept_h = (DEPT*)my_malloc(sizeof(DEPT));
temp_d = dept_h;
for(i=0; i < n_depts; i++){
        fscanf(fp, "%*s %d", &dept_no_t);
        fscanf(fp, "%*s %d %*s %d", &nmachines_t, &nmach_in_use_t);
        fscanf(fp, "%*s %d %*s %d", &row_t, &col_t);
        temp_d->dept_no = dept_no_t;
        temp_d->nmachines = nmachines_t;
        temp_d->nmach_in_use = nmach_in_use_t;
        temp_d->row = row_t;
        temp_d->col = col_t;
        temp_d->mach = (MACHINE*)my_malloc(sizeof(MACHINE));
        temp_m = temp_d->mach;
        fscanf(fp, "%*s %*s %*s %*s %*s");
        for(j=0; j < nmachines_t; j++){
                fscanf(fp, "%d %d %d %d %d", &row_t, &col_t, &mach_no_t, &repeat_t, \
                        &setup_for_t);
                fscanf(fp, "%d %d %d", &mstate_t, &user_stopped_t, &state_change_time_t);
                fscanf(fp, "%d %d %d", &idle_t, &setup_t, &busy_t);
                temp_m->row = row_t;
                temp_m->col = col_t;
                temp_m->mach_no = mach_no_t;
                temp_m->repeat = repeat_t;
                temp_m->setup_for = setup_for_t;
                map_part_type(temp_m->setup_for, &pchar);
                temp_m->setup_for_in_char = pchar;
                temp_m->mstate = mstate_t;
                temp_m->user_stopped = user_stopped_t;
                temp_m->state_change_time = (long)state_change_time_t;
                temp_m->time_idle = (long)idle_t;
                temp_m->time_setup = (long)setup_t;
                temp_m->time_busy = (long)busy_t;
                if(j == nmachines_t-1){
```

```
                        temp_m->next = NULL;
                }else{
                        temp_m->next = (MACHINE*)my_malloc(sizeof(MACHINE));
                        temp_m = temp_m->next;
                }
        }
        if(i == n_depts-1){
                temp_d->next = NULL;
        }else{
                temp_d->next = (DEPT*)my_malloc(sizeof(DEPT));
                temp_d = temp_d->next;
        }
}
temp_m = NULL;
temp_d = NULL;

temp_d = dept_h;
fscanf(fp, "%*s %*s");
for(i=0; i < n_depts; i++){
        temp_d->part_info = (PART_INFO*)my_malloc(sizeof(PART_INFO));
        temp_pi = temp_d->part_info;
        if(i == 0){
                fscanf(fp, "%d", &dept_no_t);
        }

        for(j=1; dept_no_t == i+1 ; j++){
                fscanf(fp, "%d %d %d", &row_t, &col_t, &repeat_t);
                fscanf(fp, "%d %*c %d %d", &ptype_t, &pre_t, &post_t);
                temp_pi->row = row_t;
                temp_pi->col = col_t;
                temp_pi->repeat = repeat_t;
                temp_pi->ptype = ptype_t;
                map_part_type(temp_pi->ptype, &pchar);
                temp_pi->ptype_in_char = pchar;
                temp_pi->pre = pre_t;
                temp_pi->post = post_t;
                temp_pi->dept_no = dept_no_t;
                if(dept_no_t == n_depts && j == count){
                        dept_no_t = dept_no_t + 1;
                }else{
                        fscanf(fp, "%d", &dept_no_t);
                }
                if(dept_no_t != i+1){
                        temp_pi->next = NULL;
                }else{
                        temp_pi->next = (PART_INFO*)my_malloc(sizeof(PART_INFO));
                        temp_pi = temp_pi->next;
                }
        }
        temp_d = temp_d->next;
}
```

```
temp_pi = NULL;
temp_d = NULL;

fscanf(fp, "%*s %*s");
raw_mat = (RAW*)my_malloc(sizeof(RAW));
temp_raw = raw_mat;
for(i=0; i < n_raw_mat; i++){
        fscanf(fp, "%d %d %d %d", &raw_mat_no, &cost, &row_t, &col_t);
        temp_raw->raw_mat_no = raw_mat_no;
        temp_raw->cost = cost;
        temp_raw->row = row_t;
        temp_raw->col = col_t;
        if(i == n_raw_mat-1){
                temp_raw->next = NULL;
        }else{
                temp_raw->next = (RAW*)my_malloc(sizeof(RAW));
                temp_raw = temp_raw->next;
        }
}
temp_raw = NULL;

fscanf(fp, "%*s %*s");
product = (PRODUCT*)my_malloc(sizeof(PRODUCT));
temp_prod = product;
for(i=0; i < n_products; i++){
        fscanf(fp, "%d %d %d", &prod_no, &price, &qnty_produced);
        fscanf(fp, "%d %d %d", &limit, &row_t, &col_t);
        temp_prod->prod_no = prod_no;
        temp_prod->price = price;
        temp_prod->qnty_produced = qnty_produced;
        temp_prod->limit = limit;
        temp_prod->row = row_t;
        temp_prod->col = col_t;
        if(i == n_products-1){
                temp_prod->next = NULL;
        }else{
                temp_prod->next = (PRODUCT*)my_malloc(sizeof(PRODUCT));
                temp_prod = temp_prod->next;
        }
}
temp_prod = NULL;

fscanf(fp, "%*s");
for(i=0; i < 2; i++){
        fscanf(fp, "%*s %*d");
        for(j=0; j < n_depts; j++){
                for(k=0; k < n_ptypes; k++){
                        fscanf(fp, "%d", &stime[i][j][k]);
                }
        }
}
```

```
fscanf(fp, "%*s");
for(i=0; i < 2; i++){
        fscanf(fp, "%*s %*d");
        for(j=0; j < n_depts; j++){
                for(k=0; k < n_ptypes; k++){
                        fscanf(fp, "%d", &mtime[i][j][k]);
                }
        }
}

fscanf(fp, "%*s %*s %*s");
for(i=0; i < n_ptypes; i++){
        fscanf(fp, "%d", &ptype_t);
        fscanf(fp, "%d", &raw_for_ptype[ptype_t-1]);
}

fscanf(fp, "%*s %d", &temp_time_now);
time_now = (long)temp_time_now;
fscanf(fp, "%*s %d", &week_now);
fscanf(fp, "%*s %d", &day_now);
fscanf(fp, "%*s %d", &hr_now);
fscanf(fp, "%*s %d", &min_now);
fscanf(fp, "%*s %d", &deleted);
fscanf(fp, "%*s %d", &user_specified);
fscanf(fp, "%*s %d", &processing);
fscanf(fp, "%*s %d", &auto_purchase);
fscanf(fp, "%*s %d", &auto_setup);
fscanf(fp, "%*s %d", &auto_move);
fscanf(fp, "%*s %d", &zoom_row);
fscanf(fp, "%*s %d", &zoom_col);
fscanf(fp, "%*s %d", &zoom_dept);
fscanf(fp, "%*s %d", &min_inv);
fscanf(fp, "%*s %d", &purchase_lot_size);
fscanf(fp, "%*s %d", &transfer_batch);
fscanf(fp, "%*s %d", &mat_handling_cost);
fscanf(fp, "%*s %d", &other_op_expense);

stats = (STATS*)ay_malloc(sizeof(STATS));
fscanf(fp, "%*s");
fscanf(fp, "%*s %d", &stats->fin_row);
fscanf(fp, "%*s %d", &stats->fin_col);
fscanf(fp, "%*s %d", &starting_cash);
stats->start_cash = (long)starting_cash;
fscanf(fp, "%*s %d", &now_cash);
stats->cash_now = (long)now_cash;
fscanf(fp, "%*s %d", &revenue);
stats->sales_revenue = (long)revenue;
fscanf(fp, "%*s %d", &stats->raw_mat_exp);
fscanf(fp, "%*s %d", &stats->mat_hand_exp);
fscanf(fp, "%*s %d", &stats->other_op_exp);
```

```
    fscanf(fp, "%*s %d", &stats->net_cash_flow);
    fscanf(fp, "%*s %d", &temp_event_time);
    event_time = (long)temp_event_time;
    fscanf(fp, "%*s %d", &week);
    fscanf(fp, "%*s %d", &day);
    fscanf(fp, "%*s %d", &hr);
    fscanf(fp, "%*s %d", &min);
    fscanf(fp, "%*s %d", &event_var1);
    fscanf(fp, "%*s %d", &event_var2);
    fscanf(fp, "%*s %d", &event_var3);
    fscanf(fp, "%*s %d", &event_var4);
    fscanf(fp, "%*s %s", event_type);
    fscanf(fp, "%*s %ls", modify);
    modifier = modify[0];
    read_cal_list(fp);
    fclose(fp);
    return(0);
}
```

```
#include <stdio.h>
#include <conio.h>
#include "user\std.h"
#include "user\vi.h"
#include "user\keydefs.h"
#include "user\opt.h"

main()
{
     char filename[15];
     int k, delay_var, delay;
     unsigned long int sim_maxtime = 14400L;
     char ch, optionch;
     int break_point, break_time;
     int prev_zoom, option;
     int end_of_week, reset_time;

     state = FALSE;
     system("cls");
     printf("\n %s \n","1. Start simulation from beginning");
     printf("\n %s \n \n","2. Start from a previously saved simulation state");
     printf("%s", "option: ");
back:
     optionch = getch();
     switch(optionch){
            case '1':
            case '2':
                           option = optionch - '0';
                           putc(optionch,stderr);
                           break;
            default:
                           beep();
                           goto back;
     }

     if(option == 2){
            printf("\n\n");
            printf("Enter saved state filename[.OUT]: ");
            scanf("%s", filename);
            read_sim_state(filename);
            state = TRUE;
            goto front;
     }
     printf("\n\n");
     printf("Enter filename[.SIM]: ");
     scanf("%s", filename);
     if(read_data(filename) != 0){
            printf("fatal error in file read\n");
            exit(0);
     }
front:
```

```
delay_var = 2;
break_time = 0;
end_of_week = 0;
reset_time = 0;
if(state == FALSE){
        time_now = 0L;
        week_now = day_now = hr_now = min_now = 0;
        pace = 5;
        zoom_dept = 1;
        zoom_row = 4;
        zoom_col = 39;
        processing = FALSE;
}
freezed = 1;
initialize();
init_stat();
set_screen();
if(state == FALSE){
        form_calendar();
}
set_cal_window();
display_calendar();
zoom(zoom_dept);
set_zoom_scr();
query();
print_modes();
break_point = MIN_PER_DAY;
update_stat_inv_info();
init_on_hand();
if(auto_purchase && state == FALSE){
        check_inventory();
}
if(auto_setup && state == FALSE){
        initial_setup();
}
get_next_event();
update_pace();
while(time_now <= sim_maxtime){
        if(!freezed){
                while(event_time == time_now){
                        process_event();
                        get_next_event();
                }
                if(pageno == 1){
                        update_zoom_act(zoom_dept);
                }
                if(pageno == 3){
                        update_act_report();
                }
                check_free_machs();
                update_stat_inv_info();
```

```
}
for(delay=0; delay < delay_var; delay++){
        if (kbhit()){
                k = getkey();
                switch(k){
                        case 'e':
                        case 'E':
                        case K_ALTE:
                                if(pageno == 2){
                                        edit_calendar();
                                }
                                break;
                        case 'f':
                        case 'F':
                        case K_ALTF:
                                if(!freezed)
                                        freeze();
                                break;
                        case 'r':
                        case 'R':
                        case K_ALTR:
                                viblank(0, 23, 0, 23, 79);
                                if(freezed)
                                        unfreeze();
                                if(reset_time){
                                        reset_time = 0;
                                        reset_finance();
                                }
                                if(break_time == 1){
                                        break_time = 0;
                                        viblank(0, 23, 0, 23, 79);
                                        viblank(1, 23, 0, 23, 79);
                                        viblank(2, 23, 0, 23, 79);
                                        viblank(3, 23, 0, 23, 79);
                                }
                                break;
                        case 'q':
                        case 'Q':
                        case K_ALTQ:
                                if(!freezed) stop_sim();
                                viblank(pageno, 23, 0, 23, 79);
                                viprintf(pageno, 23, 30, B_BLACK+F_RED+INT, "%s",\
                                         "Are you sure? (y/n)");
                                ch = getch();
                                if((ch == 'y') || (ch == 'Y')){
                                        wind_up();
                                        exit(0);
                                }else{
                                        viblank(pageno, 23, 0, 23, 79);
                                        if(!freezed) start_sim();
                                }
```

```
        break;
case 'b':
case 'B':
case K_ALTB:
        if(pageno != 0){
                set_page(0);
        }
        if(!freezed){
                stop_sim();
        }
        break_point = get_break_point(break_point);
        if(!freezed){
                start_sim();
        }
        if(pageno != 0){
                set_page(pageno);
        }
        break;
case K_ALTM:
        if(pageno == 0){
                if(!freezed){
                        stop_sim();
                }
                get_modes();
                if(!freezed){
                        start_sim();
                }
        }
        break;
case 's':
case 'S':
case K_ALTS:
        if(pageno == 0){
                if(!freezed){
                        stop_sim();
                }
                user_schedule();
                if(!freezed){
                        start_sim();
                }
        }
        break;
case 'c':
case 'C':
case K_ALTC:
        if(pageno == 1){
                if(!freezed){
                        stop_sim();
                }
                change_vars();
                if(!freezed){
```

```
                        start_sim();
                }
        }
        break;
case 'p':
case 'P':
case K_ALTP:
        if(!freezed)
                freeze();
        print_scr(pageno);
        unfreeze();
        break;
case K_F1:
        pageno = 0;
        set_page(pageno);
        break;
case K_F2:
        if(pageno == 1){
                prev_zoom = zoom_dept;
                zoom_dept = get_dept_no(prev_zoom);
                if(zoom_dept != prev_zoom){
                        zoom(zoom_dept);
                }
        }else{
                update_zoom_clock();
                update_zoom_act(zoom_dept);
                pageno = 1;
                set_page(pageno);
        }
        break;
case K_F3:
        pageno = 2;
        set_page(pageno);
        break;
case K_F4:
        update_stat_clock();
        update_stat_finance();
        update_stat_prod_info();
        update_act_report();
        pageno = 3;
        set_page(pageno);
        break;
case '+':
        if(delay_var >= 102){
                delay_var -= 100;
                pace++;
                update_pace();
        }
        break;
case '-':
        if(delay_var <= 302){
```

```
                                        delay_var += 100;
                                        pace--;
                                        update_pace();
                                }
                                break;
                        case K_PGUP:
                                if(pageno == 2){
                                        set_cal_pgup();
                                }
                                break;
                        case K_PGDN:
                                if(pageno == 2){
                                        set_cal_pgdn();
                                }
                                break;
                        case K_UP:
                                if(pageno == 2){
                                        set_cal_1_up();
                                }
                                break;
                        case K_DOWN:
                                if(pageno == 2){
                                        set_cal_1_down();
                                }
                                break;
                        default:
                                break;

                }
        }
}
if(end_of_week){
        reset_time = 1;
        end_of_week = 0;
        calc_cash_flow();
        reset_quantity();
        draw_products();
}
if(break_time){
        if(!freezed){
                freeze();
                viblank(pageno, 23, 0, 23, 79);
                viprintf(pageno, 23, 24, MSG_COLOR, "%s",\
                        "Break point: Type 'R' to continue");
        }
}
if(((event_time) % (unsigned long int)break_point) == 0){
        break_time = 1;
}
if(((event_time) % MIN_PER_WEEK) == 0){
        end_of_week = 1;
```

```
            }
            if(!freezed){
                    process_event();
                    if(auto_purchase){
                            check_inventory();
                    }
                    get_next_event();
            }
    }
    pageno = 0;
    set_page(pageno);
    wind_up();
}
```

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "user\bioslib.h"
#include "user\keydefs.h"
#include "user\std.h"
#include "user\opt.h"
#include "user\vi.h"

#define SPACES              "          "
#define FORMFEED     '\x0C'

int print_init(void);
int print_scr(int);

/* Function to initialize the printer */
int print_init()
{
     prt_init(0);
}

/* Function to dump the screen */
int print_scr(header)
int header;
{
     char buf[80];
     int i, print_abort, k;
     char ch;
     union REGS inregs, outregs;

     print_abort = FALSE;
     if((header < 0) || (header > 3)) return(1);
     viblank(pageno, 23, 0, 23, 79);
     viprintf(pageno, 23, 15, MSG_COLOR, "%s",\
            "Make sure printer is on and press a key to continue");
     ch = getch();
     viblank(pageno, 23, 0, 23, 79);
     viprintf(pageno, 23, 35, MSG_COLOR+BLNK, "%s", "Printing...");
     for(i=0; i < 80; i++){
            buf[i] = ' ';
     }
     switch(header){
            case 0:
                   sprintf(buf, "%s%s%s%s%c%c%c%c", SPACES, SPACES, SPACES,\
                          "SNAPSHOT", '\n', '\n', '\n', '\0');
                   break;
            case 1:
                   sprintf(buf, "%s%s%s%s%c%c%c%c", SPACES, SPACES, SPACES,\
                          "ZOOM REPORT", '\n', '\n', '\n', '\0');
                   break;
            case 2:
```

```
                sprintf(buf, "%s%s%s%s%c%c%c%c", SPACES, SPACES, SPACES,\
                        "CALENDAR LISTING", '\n', '\n', '\n', '\0');
                break;
        case 3:
                sprintf(buf, "%s%s%s%s%c%c%c%c", SPACES, SPACES, SPACES,\
                        "STATISTICS REPORT", '\n', '\n', '\n', '\0');
                break;
        default:
                break;
    }
    for(i=0; !(print_abort) && i < 80 && buf[i] != '\0'; i++){
        if((prt_print(0, buf[i]) & 1) != 0){
                beep();
                viblank(pageno, 23, 0, 23, 79);
                viprintf(pageno, 23, 10, MSG_COLOR, "%s",\
                        "Printer not ready:\
                         press any key to try again or ESC to abort");
                ch = getch();
                if(ch == K_ESC){
                        print_abort = TRUE;
                }else{
                        i--;
                        viblank(pageno, 23, 0, 23, 79);
                        viprintf(pageno, 23, 35, MSG_COLOR+BLNK, "%s",\
                                "Printing...");
                }
        }
    }
    if(!print_abort){
        int86(PRINT_SCRN, &inregs, &outregs);
        prt_print(0, FORMFEED);
    }
    viblank(pageno, 23, 0, 23, 79);
    return(0);
}
```

```
#include <stdio.h>
#include "user\vi.h"
#include "user\opt.h"
#include "user\process.h"
#include "user\std.h"

#define dim(x)        (sizeof(x)/sizeof(x[0]))        /* macro to return the number
                                                      of elements in a structure */

/* --- Functions to process events --- */
int process_event(void);
int check_inventory(void);
int check_free_machs(void);
int print_errorp(void);
int print_errord(void);
int print_errorm(void);
int initial_setup(void);
void update_pace(void);
void update_finance(void);
void update_stat_finance(void);
void update_stat_prod_info(void);
void update_act_report(void);
void update_zoom_clock(void);
void update_stat_clock(void);
int print_skip(int);
int mach_proc_ptype(DEPT*, int, int);
int update_zoom_window(int, int, int, char);
void update_stat_inv_info(void);
void init_on_hand(void);

/* --- Private functions defined in process.c --- */
static void update_scr_clk(void);
static void update_part_info(PART_INFO*);
static void update_pre_info(PART_INFO*);
static void update_post_info(PART_INFO*);
static void update_prod_info(PRODUCT*);
static void update_clock(void);
static void update_inv_finance(),
static int count[10][3];
static void update_on_hand(PART_INFO*);

/* Procedure to advance simulation clock */
int advance_clock(dummy1, dummy2, dummy3, dummy4)
int dummy1, dummy2, dummy3, dummy4;
{
    time_now = event_time;
    week_now = week;
    day_now  = day;
    hr_now   = hr;
    min_now  = min;
    update_scr_clk();
```

```
    schedule(CLOCK_TICK, time_now+5L, 0, 0, 0, 0);
}

/* Function to update the clock */
static void update_clock()
{
    time_now = event_time;
    week_now = week;
    day_now  = day;
    hr_now   = hr;
    min_now  = min;
    update_scr_clk();
}

/* Function to update the screen clock */
static void update_scr_clk()
{
    viprintf(0, 1, 1, CLOCK_COLOR, "%2d", week_now);
    if(week_now < 10) viprintf(0, 1, 1, CLOCK_COLOR, "%c", '0');
    viprintf(0, 1, 7, CLOCK_COLOR, "%1d", day_now);
    viprintf(0, 4, 2, CLOCK_COLOR, "%2d:%2d", hr_now, min_now);
    if(min_now < 10) viprintf(0, 4, 5, CLOCK_COLOR, "%c", '0');
    if(pageno == 1){
        update_zoom_clock();
    }
    if(pageno == 3){
        update_stat_clock();
    }
}

/* Function to update the clock in statistics screen */
void update_stat_clock()
{
    viprintf(3, 0, 11, CLOCK_COLOR, "%2d", week_now);
    if(week_now < 10){
        viprintf(3, 0, 11, CLOCK_COLOR, "%c", '0');
    }
    viprintf(3, 0, 20, CLOCK_COLOR, "%1d", day_now);
    viprintf(3, 0, 69, CLOCK_COLOR, "%2d:%2d", hr_now, min_now);
    if(min_now < 10){
        viprintf(3, 0, 72, CLOCK_COLOR, "%c", '0');
    }
}

/* Function to update the clock in the zoom window */
void update_zoom_clock()
{
    viprintf(1, 1, 9, CLOCK_COLOR, "%2d", week_now);
    if(week_now < 10){
        viprintf(1, 1, 9, CLOCK_COLOR, "%c", '0');
    }
```

```
    viprintf(1, 1, 18, CLOCK_COLOR, "%1d", day_now);
    viprintf(1, 3, 11, CLOCK_COLOR, "%2d:%2d", hr_now, min_now);
    if(min_now < 10){
        viprintf(1, 3, 14, CLOCK_COLOR, "%c", '0');
    }
}

/* Function to update the pace display on the screen */
void update_pace()
{
    viprintf(0, 12, 6, CLOCK_COLOR, "%1d", pace);
}

/* The Event Processor */
int process_event()
{
    int i, j;
    DEPT *temp_d;
    MACHINE *temp_m;
    int mach_no, make_idle;

    if(!deleted){
        for(i=0; i < dim(cal_fncs); i++){
            j = strcmp(cal_fncs[i].name, event_type);
            if(j == 0){
                (*cal_fncs[i].fnc)(event_var1, event_var2,\
                                   event_var3, event_var4);
                del_event_from_cal();
                return(0);
            }
        }
    }else{
        if(strcmp(event_type, LOAD) == 0){
            for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != event_var2;\
                temp_d = temp_d->next);
            if(temp_d == NULL) print_errord();
            if(event_var1 == 0){
                for(temp_m = temp_d->mach; temp_m != NULL &&\
                                !(temp_m->setup_for == event_var4 && \
                                  temp_m->repeat == event_var3);\
                                      temp_m = temp_m->next);
                if(temp_m == NULL) print_errorm();
                if(temp_m->mstate == 1){
                    mach_no = temp_m->mach_no;
                    make_idle = 1;
                }else{
                    make_idle = 0;
                }
            }else{
                mach_no = event_var1;
            }
```

```
                if(make_idle){
                        for(temp_m = temp_d->mach; temp_m != NULL &&\
                            temp_m->mach_no != mach_no; temp_m = temp_m->next);
                        if(temp_m == NULL) print_errorm();
                        if((temp_m->mstate == 1) && ((temp_m->repeat == event_var3) && \
                             (temp_m->setup_for == event_var4))){
                                viprintf(0, temp_m->row+1, temp_m->col+1, MACHINE_COLOR, \
                                        "%c", IDLE);
                                temp_m->time_setup += (time_now - temp_m->state_change_time);
                                temp_m->state_change_time = time_now;
                                temp_m->mstate = 0;
                                update_clock();
                                if(temp_d->dept_no == zoom_dept){
                                        update_zoom_window(temp_m->mach_no, temp_m->mstate, \
                                        temp_m->repeat, temp_m->setup_for_in_char);
                                }
                        }
                }
        }
    }
    del_event_from_cal();
    return(1);
}

/* Procedure to move the part */
int move_part(repeat, part_type, dept_no, trans_batch)
int repeat, part_type, dept_no, trans_batch;
{
    int nth_occurance, op_no, to_dept, earnings, new_repeat, i;
    DEPT *temp_d;
    PART_INFO *temp_pi;
    PRODUCT *temp_prod;
    MACHINE *temp_m;
    char pchar;
    int temp_batch_size;

    for(op_no=0, nth_occurance=0; nth_occurance != repeat+1; nth_occurance++,op_no++){
            for(;route[part_type-1][op_no] != dept_no; op_no++);
    }
    to_dept = route[part_type-1][op_no];
    for(i=0, new_repeat=0; i < op_no; i++){
            if(route[part_type-1][i] == to_dept){
                    new_repeat++;
            }
    }
    if(to_dept == 0){
            for(temp_prod=product; temp_prod->prod_no != part_type;\
                            temp_prod = temp_prod->next);
            for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no;\
                            temp_d = temp_d->next);
            for(temp_pi = temp_d->part_info; temp_pi != NULL &&\
```

```
                        !(temp_pi->ptype == part_type && temp_pi->repeat == repeat);\
                        temp_pi = temp_pi->next);
        if((temp_prod->limit-temp_prod->qnty_produced)==0){
                temp_pi->post=(temp_pi->post)+trans_batch;
                update_post_info(temp_pi);
                update_on_hand(temp_pi);
                update_clock();
                return(0);
        }
        if(trans_batch > (temp_prod->limit-temp_prod->qnty_produced)){
                temp_pi->post=(temp_pi->post)+trans_batch-(\
                   temp_prod->limit-temp_prod->qnty_produced);
                update_post_info(temp_pi);
                update_on_hand(temp_pi);
                temp_batch_size = (temp_prod->limit-temp_prod->qnty_produced);
                temp_prod->qnty_produced += temp_batch_size;
                update_prod_info(temp_prod);
                earnings = temp_prod->price*temp_batch_size;
                cash += earnings;
                stats->sales_revenue += earnings;
        }else{
                update_post_info(temp_pi);
                update_on_hand(temp_pi);
                temp_prod->qnty_produced += trans_batch;
                update_prod_info(temp_prod);
                earnings = temp_prod->price*trans_batch;
                cash += earnings;
                stats->sales_revenue += earnings;
        }
}else{
        for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no;\
                        temp_d = temp_d->next);
        for(temp_pi = temp_d->part_info; temp_pi != NULL &&\
                        !(temp_pi->ptype == part_type && temp_pi->repeat == repeat);\
                        temp_pi = temp_pi->next);
        update_post_info(temp_pi);
        for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != to_dept;\
                        temp_d = temp_d->next);
        for(temp_pi = temp_d->part_info; temp_pi != NULL &&\
                        !(temp_pi->ptype == part_type && temp_pi->repeat == new_repeat);\
                        temp_pi = temp_pi->next);
        temp_pi->pre += trans_batch;
        update_pre_info(temp_pi);

        for(temp_m = temp_d->mach; temp_m != NULL &&\
                        !(temp_m->repeat == new_repeat && temp_m->setup_for == part_type);\
                        temp_m = temp_m->next);
        if(temp_m != NULL){
                /* There is a machine running this part_type */
                if((temp_m->mstate == 0) && (temp_pi->pre > 0)){
                        /* That machine is idle and if pre is not empty */
```

```
                        /* This case is taken care of by check_free_machs() */
                }
        }else{
                /* There are no machines running this part_type */
                if(temp_d->nmach_in_use < temp_d->nmachines){
                        /* And there are some unused machines */
                        if(auto_setup){
                                for(temp_m = temp_d->mach; temp_m != NULL && \
                                        temp_m->setup_for != UN_UTILIZED;\
                                                temp_m = temp_m->next);
                                if(temp_m == NULL) viprintf(0, 23, 0, MSG_COLOR,\
                                                        "%s", "check nmach_in_use");
                                temp_m->repeat = new_repeat;
                                temp_m->setup_for = part_type;
                                map_part_type(temp_m->setup_for, &pchar);
                                temp_m->setup_for_in_char = pchar;
                                temp_m->state_change_time = time_now;
                                temp_m->mstate = 1;
                                schedule(SETUP, time_now+IL, temp_m->mach_no,\
                                                temp_d->dept_no, temp_m->repeat,\
                                                temp_m->setup_for);
                                temp_d->nmach_in_use++;
                        }
                }
        }
    }
    expense += MAT_HANDLING_COST;
    cash -= MAT_HANDLING_COST;
    stats->cash_now = cash;
    stats->mat_hand_exp += MAT_HANDLING_COST;
    update_finance();
    update_clock();
    return(0);
}

/* Procedure to load the machine */
int load_machine(mach_no, dept_no, repeat, part_type)
int mach_no, dept_no, repeat, part_type;
{

    DEPT *temp_d;
    MACHINE *temp_m;
    PART_INFO *temp_pi;
    int n_pre;

    for(temp_d = dept_h; temp_d != NULL && *temp_d->dept_no != dept_no;\
                                        temp_d = temp_d->next);
    if(temp_d == NULL) print_errord();

    if(mach_no != 0){
        for(temp_m = temp_d->mach;\
```

```
                temp_m != NULL && temp_m->mach_no != mach_no;\
                temp_m = temp_m->next);

        if(temp_m == NULL) print_skip(1);
        for(temp_pi = temp_d->part_info; temp_pi != NULL &&\
                !(temp_pi->ptype == part_type && temp_pi->repeat == repeat);\
                                        temp_pi = temp_pi->next);
        if(temp_pi == NULL) print_skip(2);
        if(temp_pi->pre < 0) print_skip(3);
        if(temp_pi->pre == 0){
                temp_m->time_setup += (time_now - temp_m->state_change_time);
                temp_m->state_change_time = time_now;
                temp_m->mstate = 0;
                viprintf(0, temp_m->row+1, temp_m->col+1, MACHINE_COLOR, "%c", IDLE);
                update_clock();
                if(temp_d->dept_no == zoom_dept){
                        update_zoom_window(temp_m->mach_no, temp_m->mstate, \
                                        temp_m->repeat, temp_m->setup_for_in_char);
                }
                return(0);
        }
        if(temp_m->mstate == 0){
                temp_m->time_idle += (time_now - temp_m->state_change_time);
                temp_m->state_change_time = time_now;
        }
        if(temp_m->mstate == 1){
                temp_m->time_setup += (time_now - temp_m->state_change_time);
                temp_m->state_change_time = time_now;
        }
        temp_m->mstate = 2;
        viprintf(0, temp_m->row+1, temp_m->col+1, MACHINE_COLOR+BLNK, "%c", PROCESSING);
        if(temp_d->dept_no == zoom_dept){
        update_zoom_window(temp_m->mach_no, temp_m->mstate, temp_m->repeat, temp_m->setup_for_in_char);
        }
        schedule(UNLOAD, time_now+(unsigned long int)mtime[repeat][dept_no-1][part_type-1],\
                                        temp_m->mach_no, dept_no, 0, 0);
        n_pre = nmach_proc_ptype(temp_d, repeat, part_type);
        if(temp_pi->pre > n_pre*2-1){
        schedule(LOAD, time_now+(unsigned long int)mtime[repeat][dept_no-1][part_type-1],\
                                        mach_no, dept_no, repeat, part_type);
        }
}else{
        for(temp_m = temp_d->mach;\
                        temp_m != NULL  &&\
                        !(temp_m->setup_for == part_type &&\
                        temp_m->repeat == repeat &&\
                        temp_m->mstate == 0);\
                        temp_m = temp_m->next);
        if(temp_m == NULL) print_skip(1);

        for(temp_pi = temp_d->part_info; temp_pi != NULL &&\
```

```
                !(temp_pi->ptype == part_type && temp_pi->repeat == repeat);\
                                        temp_pi = temp_pi->next);
        if(temp_pi == NULL) print_skip(2);
        if(temp_pi->pre < 0) print_skip(3);
        if(temp_pi->pre == 0){
                temp_m->time_setup += (time_now - temp_m->state_change_time);
                temp_m->state_change_time = time_now;
                temp_m->mstate = 0;
                viprintf(0, temp_m->row+1, temp_m->col+1, MACHINE_COLOR, "%c", IDLE);
                update_clock();
                if(temp_d->dept_no == zoom_dept){
                        update_zoom_window(temp_m->mach_no, temp_m->mstate, \
                                temp_m->repeat, temp_m->setup_for_in_char);
                }
                return(0);
        }
        if(temp_m->mstate == 0){
                temp_m->time_idle += (time_now - temp_m->state_change_time);
                temp_m->state_change_time = time_now;
        }
        if(temp_m->mstate == 1){
                temp_m->time_setup += (time_now - temp_m->state_change_time);
                temp_m->state_change_time = time_now;
        }
        temp_m->mstate = 2;
        viprintf(0, temp_m->row+1, temp_m->col+1, MACHINE_COLOR+BLNK, "%c", PROCESSING);
        if(temp_d->dept_no == zoom_dept){
             update_zoom_window(temp_m->mach_no, temp_m->mstate, temp_m->repeat, \
                                temp_m->setup_for_in_char);
        }
        schedule(UNLOAD, time_now+(unsigned long int)mtime[repeat][dept_no-1][part_type-1],\
                                        temp_m->mach_no, dept_no, 0, 0);
        n_pre = nmach_proc_ptype(temp_d, repeat, part_type);
        if(temp_pi->pre > n_pre*2-1){
        schedule(LOAD, time_now+(unsigned long int)mtime[repeat][dept_no-1][part_type-1],\
                                        mach_no, dept_no, repeat, part_type);
        }
    }
    update_clock();
    return(0);
}

/* Procedure to unload the machine */
int unload_machine(mach_no, dept_no, dummy1, dummy2)
int mach_no, dept_no, dummy1, dummy2;
{
    DEPT *temp_d;
    MACHINE *temp_m;
    PART_INFO *temp_pi;
    int repeat, part_type;
    int nth_occurance, op_no, to_dept, new_repeat, i;
```

```
    for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no;\
                        temp_d = temp_d->next);
    if (temp_d == NULL) print_errord();
    for(temp_m = temp_d->mach; temp_m != NULL && temp_m->mach_no != mach_no;\
                        temp_m = temp_m->next);
    if(temp_m == NULL) print_errorm();
    repeat = temp_m->repeat;
    part_type = temp_m->setup_for;
    for(temp_pi = temp_d->part_info; temp_pi != NULL &&\
                !(temp_pi->ptype == part_type && temp_pi->repeat == repeat);\
                        temp_pi = temp_pi->next);
    if(temp_pi == NULL) print_errorp();
    temp_pi->pre--;
    temp_pi->post++;
    update_part_info(temp_pi);
    temp_m->time_busy += (time_now - temp_m->state_change_time);
    temp_m->state_change_time = time_now;
    temp_m->mstate = 0;
    viprintf(0, temp_m->row+1, temp_m->col+1, MACHINE_COLOR, "%c", IDLE);
    update_clock();
    if(temp_d->dept_no == zoom_dept){
            update_zoom_window(temp_m->mach_no, temp_m->mstate, temp_m->repeat, \
                        temp_m->setup_for_in_char);
    }
    if(auto_move && (temp_pi->post == TRANSFER_BATCH)){
            schedule(MOVE, time_now,\
                    repeat, part_type, dept_no, TRANSFER_BATCH);
            temp_pi->post -= TRANSFER_BATCH;
    }

    if(auto_move == FALSE){
            for(op_no=0,nth_occurance=0;nth_occurance!=repeat+1;nth_occurance++,op_no++){
                    for(;route[part_type-1][op_no] != dept_no; op_no++);
            }
            to_dept = route[part_type-1][op_no];
            for(i=0, new_repeat=0; i < op_no; i++){
                    if(route[part_type-1][i] == to_dept){
                            new_repeat++;
                    }
            }
            if(to_dept == 0){
                    update_on_hand(temp_pi);
            }
    }
    return(0);
}

/* procedure to purchase raw material */
int purchase_rawmat(raw_mat_no, part_type, qnty, dummy)
int raw_mat_no, part_type, qnty, dummy;
```

```
{
    int cost, dept_no;
    DEPT *temp_d;
    PART_INFO *temp_pi;
    RAW *temp_raw;

    for(temp_raw = raw_mat;\
        temp_raw != NULL && temp_raw->raw_mat_no != raw_mat_no;\
            temp_raw = temp_raw->next);
    if(temp_raw == NULL) print_errorp();
    cost = temp_raw->cost*qnty;
    expense += cost;
    cash -= cost;
    stats->cash_now = cash;
    stats->raw_mat_exp += cost;
    update_finance();
    dept_no = route[part_type-1][0];
    for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no;\
                                        temp_d = temp_d->next);
    if(temp_d == NULL) print_errord();
    for(temp_pi = temp_d->part_info;\
            temp_pi != NULL && temp_pi->ptype != part_type;\
                                    temp_pi = temp_pi->next);
    if(temp_pi == NULL) print_errorp();
    if(modifier == USER_SPECIFIED){
        temp_pi->pre += qnty;
    }
    /* display the updated contents of pre */
    update_pre_info(temp_pi);
    update_clock();
    return(0);
}

/* Function to print error message if something is wrong in part_info */
int print_errorp()
{
    if(pageno > 0){
        pageno = 0;
        set_page(pageno);
    }
    viprintf(0, 22, 0, NORM, "%s", "Error in part_info");
    exit(0);
}

/* Function to warn about unprocessed events */
int print_skip(number)
int number;
{
    if(pageno > 0){
        pageno = 0;
        set_page(pageno);
```

```
    }
    if(number == 1){
            viprintf(0, 22, 0, NORM, "%s", "No machine free..skipping one load");
            exit(0);
    }
    if(number == 2){
            viprintf(0, 22, 0, NORM, "%s", "Error in part_info..skipping load");
            exit(0);
    }
    if(number == 3){
            beep();
            viprintf(0, 22, 30, NORM, "%s", "No part in pre..skipping one load");
    }
}

/* Function to print error message if something is wrong in department structure */
int print_errord()
{
    if(pageno > 0){
            pageno = 0;
            set_page(pageno);
    }
    viprintf(0, 22, 0, NORM, "%s", "Error in dept list");
    exit(0);
}

/* Function to print error message if something is wrong in machine structure */
int print_errorm()
{
    if(pageno > 0){
            pageno = 0;
            set_page(pageno);
    }
    viprintf(0, 22, 20, NORM, "%s", "Error in machine list");
    exit(0);
}

/* Procedure to setup machine */
int setup_machine(mach_no, dept_no, repeat, part_type)
int mach_no, dept_no, repeat, part_type;
{
    DEPT *temp_d;
    MACHINE *temp_m;
    PART_INFO *temp_pi;
    char pchar;

    for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no;\
                                        temp_d = temp_d->next);
    if(temp_d == NULL) print_errord();
    for(temp_m = temp_d->mach; temp_m != NULL && temp_m->mach_no != mach_no;\
                                        temp_m = temp_m->next);
```

```
    if(temp_m == NULL) print_errorm();
    if(modifier == USER_SPECIFIED){
            temp_m->repeat = repeat;
            temp_m->setup_for = part_type;
            map_part_type(part_type, &pchar);
            temp_m->setup_for_in_char = pchar;
            if(temp_m->mstate == 0){
                    temp_m->time_idle += (time_now - temp_m->state_change_time);
                    temp_m->state_change_time = time_now;
            }
            if(temp_m->mstate == 2){
                    temp_m->time_busy += (time_now - temp_m->state_change_time);
                    temp_m->state_change_time = time_now;
            }
            temp_m->mstate = 1;
    }
    if(repeat > 0){
            viprintf(0, temp_m->row+1, temp_m->col-1, B_BLUE+F_GREEN+INT, "%1d", repeat);
    }else{
            viprintf(0, temp_m->row+1, temp_m->col-1, B_BLUE+F_GREEN+INT, "%c", ' ');
    }
    viprintf(0, temp_m->row+1, temp_m->col, MACHINE_COLOR, "%c", temp_m->setup_for_in_char);
    viprintf(0, temp_m->row+1, temp_m->col+1, MACHINE_COLOR+BLNK, "%c", SETTING_UP);
    if(temp_d->dept_no == zoom_dept){
            update_zoom_window(temp_m->mach_no, temp_m->mstate, temp_m->repeat, \
                               temp_m->setup_for_in_char);
    }
    for(temp_pi = temp_d->part_info;\
            temp_pi != NULL && !((temp_pi->ptype == part_type) &&\
                    (temp_pi->repeat == repeat));\
                            temp_pi = temp_pi->next);
    if(temp_pi == NULL) print_errorp();
    schedule(LOAD, time_now+(unsigned long int)stime[repeat][dept_no-1][part_type-1],\
                            mach_no, dept_no, repeat, part_type);
    update_clock();
    return(0);
}

/* Function to check inventory */
int check_inventory()
{
    DEPT *temp_d;
    PART_INFO *temp_pi;
    int i, dept_no;

    for(i=0; i < n_ptypes; i++){
            dept_no = route[i][0];
            for(temp_d = dept_h;\
                    temp_d != NULL && temp_d->dept_no != dept_no;\
                            temp_d = temp_d->next);
            for(temp_pi = temp_d->part_info;\
```

```
                temp_pi != NULL && temp_pi->ptype != (i+1);\
                        temp_pi = temp_pi->next);
        if(temp_pi->pre <= MIN_INV){
                temp_pi->pre += PURCHASE_LOT_SIZE;
                schedule(PURCHASE, time_now+1L,\
                        raw_for_ptype[i], i+1, PURCHASE_LOT_SIZE, 0);
        }
    }
}

/* Function to update cash and expense on the screen */
void update_finance()
{
    viprintf(0, 7, 1, CASH_COLOR, "%61d", cash);
    if(expense != 0){
        viprintf(0,10, 1, EXPENSE_COLOR, "%61d", expense);
    }else{
        viprintf(0,10, 1, EXPENSE_COLOR, "%s", "      ");
    }
    if(pageno == 3){
        update_stat_finance();
    }
}

/* Function to update Financial Report in Statistics window */
void update_stat_finance()
{
    int row, col;

    row = stats->fin_row;
    col = stats->fin_col;
    viprintf(3, row-1, col, STAT_FIN_COLOR, "%71d", stats->start_cash);
    viprintf(3, row, col, STAT_FIN_COLOR, "%71d", stats->cash_now);
    viprintf(3, row+2, col, STAT_FIN_COLOR, "%7d", stats->raw_mat_exp);
    viprintf(3, row+3, col, STAT_FIN_COLOR, "%7d", stats->mat_hand_exp);
    viprintf(3, row+4, col, STAT_FIN_COLOR, "%7d", stats->other_op_exp);
    viprintf(3, row+6, col, STAT_FIN_COLOR, "%71d", stats->sales_revenue);
    viprintf(3, row+7, col+1, STAT_FIN_COLOR, "%+6d", stats->net_cash_flow);
}

/* Function to update part information on the screen */
static void update_part_info(ptr_pi)
PART_INFO *ptr_pi;
{
    DEPT *ptr_dept;
    PART_INFO *tmp_pi;
    int row, col;

    if(ptr_pi->pre != 0){
        viprintf(0, ptr_pi->row, ptr_pi->col, PRE_COLOR, "%3d", ptr_pi->pre);
    }else{
```

```
            viprintf(0, ptr_pi->row, ptr_pi->col, PRE_COLOR, "%s", "   ");
        }
        if(ptr_pi->post != 0){
            viprintf(0, ptr_pi->row, ptr_pi->col+16, POST_COLOR, "%3d", ptr_pi->post);
        }else{
            viprintf(0, ptr_pi->row, ptr_pi->col+16, POST_COLOR, "%s", "   ");
        }
        if(ptr_pi->dept_no == zoom_dept){
            row = zoom_row+1;
            col = zoom_col-13;
            for(ptr_dept = dept_h; ptr_dept != NULL && ptr_dept->dept_no != zoom_dept;\
                ptr_dept = ptr_dept->next);
            if(ptr_dept == NULL) print_errorp();
            for(tmp_pi = ptr_dept->part_info; tmp_pi != NULL; tmp_pi = tmp_pi->next){
                if(tmp_pi->pre != 0){
                    viprintf(1, row, col-1, PRE_COLOR, " %3d ", tmp_pi->pre);
                }else{
                    viprintf(1, row, col-1, PRE_COLOR, "%s", "     ");
                }
                if(tmp_pi->post != 0){
                    viprintf(1, row, col+23, POST_COLOR, " %3d ", tmp_pi->post);
                }else{
                    viprintf(1, row, col+23, POST_COLOR, "%s", "     ");
                }
                row++;
            }
        }
}

/* Function to update pre information on the screen */
static void update_pre_info(ptr_pi)
PART_INFO *ptr_pi;
{
    DEPT *ptr_dept;
    PART_INFO *tmp_pi;
    int row, col;

    if(ptr_pi->pre != 0){
        viprintf(0, ptr_pi->row, ptr_pi->col, PRE_COLOR, "%3d", ptr_pi->pre);
    }else{
        viprintf(0, ptr_pi->row, ptr_pi->col, PRE_COLOR, "%s", "   ");
    }
    if(ptr_pi->dept_no == zoom_dept){
        row = zoom_row+1;
        col = zoom_col-13;
        for(ptr_dept = dept_h; ptr_dept != NULL && ptr_dept->dept_no != zoom_dept;\
            ptr_dept = ptr_dept->next);
        if(ptr_dept == NULL) print_errorp();
        for(tmp_pi = ptr_dept->part_info; tmp_pi != NULL; tmp_pi = tmp_pi->next){
            if(tmp_pi->pre != 0){
                viprintf(1, row, col-1, PRE_COLOR, " %3d ", tmp_pi->pre);
```

```
                }else{
                        viprintf(1, row, col-1, PRE_COLOR, "%s", "     ");
                }
                row++;
        }
    }
}

/* Function to update post information on the screen */
static void update_post_info(ptr_pi)
PART_INFO *ptr_pi;
{
    DEPT *ptr_dept;
    PART_INFO *tmp_pi;
    int row, col;

    if(ptr_pi->post != 0){
        viprintf(0, ptr_pi->row, ptr_pi->col+16, POST_COLOR, "%3d", ptr_pi->post);
    }else{
        viprintf(0, ptr_pi->row, ptr_pi->col+16, POST_COLOR, "%s", "   ");
    }
    if(ptr_pi->dept_no == zoom_dept){
        row = zoom_row+1;
        col = zoom_col-13;
        for(ptr_dept = dept_h; ptr_dept != NULL && ptr_dept->dept_no != zoom_dept;\
            ptr_dept = ptr_dept->next);
        if(ptr_dept == NULL) print_errorp();
        for(tmp_pi = ptr_dept->part_info; tmp_pi != NULL; tmp_pi = tmp_pi->next){
            if(tmp_pi->post != 0){
                viprintf(1, row, col+23, POST_COLOR, " %3d ", tmp_pi->post);
            }else{
                viprintf(1, row, col+23, POST_COLOR, "%s", "     ");
            }
            row++;
        }
    }
}

/* Function to update product information on the screen */
static void update_prod_info(prod_ptr)
PRODUCT *prod_ptr;
{
    viprintf(0, prod_ptr->row, prod_ptr->col+3, PRODUCT_COLOR, "%4d", \
             (prod_ptr->limit-prod_ptr->qnty_produced));
    if(pageno == 3){
        update_stat_prod_info();
    }
}

/* Function to update product information on the statistics window */
void update_stat_prod_info()
```

```
{
    int row = 16;
    int col = 16;
    PRODUCT *temp_prod;

    for(temp_prod = product; temp_prod != NULL; temp_prod = temp_prod->next, row++){
        viprintf(3, row, col, STAT_SLS_COLOR, "%3d", temp_prod->qnty_produced);
    }
}

/* Function to update Activation Report in the statistics window */
void update_act_report()
{
    unsigned long int total_time;
    unsigned long int prod_time, setup_time, idle_time;
    int prod, setup, idle;
    int row, col;
    DEPT *temp_d;
    MACHINE *temp_m;

    row = 5;
    col = 54;
    for(temp_d = dept_h; temp_d != NULL; temp_d = temp_d->next){
        for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
            total_time = time_now;
            if (total_time != 0){
                setup_time = temp_m->time_setup;
                prod_time = temp_m->time_busy;
                if(temp_m->mstate == 1){
                setup_time += (time_now - temp_m->state_change_time);
                }
                if(temp_m->mstate == 2){
                prod_time += (time_now - temp_m->state_change_time);
                }
                setup = (int)(((double)setup_time/(double)total_time)*100L);
                prod = (int)(((double)prod_time/(double)total_time)*100L);
                idle = 100-(prod+setup);
                viprintf(3, row, col, STAT_ACT_COLOR, "%3d", prod);
                viprintf(3, row, col+7, STAT_ACT_COLOR, "%3d", setup);
                viprintf(3, row, col+15, STAT_ACT_COLOR, "%3d", idle);
            }
            row++;
        }
    }
}

/* Function to schedule initial setup events */
int initial_setup()
{
    DEPT *temp_d;
    MACHINE *temp_m;
```

```
    PART_INFO *temp_pi;
    int i, dept_no;
    char pchar;

    for(i=0; i < n_ptypes; i++){
            dept_no = route[i][0];
            for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no;\
                                                    temp_d = temp_d->next);
            if(temp_d == NULL) print_errord();
            if(temp_d->nmach_in_use < temp_d->nmachines){
                    for(temp_m=temp_d->mach;temp_m!=NULL && temp_m->mstate!=0;\
                                                    temp_m = temp_m->next);

                    if(temp_m == NULL) break;
                    temp_m->repeat = 0;
                    temp_m->setup_for = i+1;
                    map_part_type(temp_m->setup_for, &pchar);
                    temp_m->setup_for_in_char = pchar;
                    temp_m->mstate = 1;
                    temp_m->state_change_time = time_now;
                    schedule(SETUP, time_now+1L, temp_m->mach_no,\
                                    temp_d->dept_no, temp_m->repeat,\
                                    temp_m->setup_for);
                    temp_d->nmach_in_use++;
            }
    }
}

/* Function to schedule load events if there are free machines */
int check_free_machs()
{
    DEPT *temp_d;
    MACHINE *temp_m;
    PART_INFO *temp_pi;
    int no_free_mach;
    int n_pre;

    for(temp_d = dept_h; temp_d != NULL; temp_d = temp_d->next){
            no_free_mach = FALSE;
            for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
                    if(temp_m->mstate == 0){
                            if(temp_m->setup_for != UN_UTILIZED){
                                    goto found_free_mach;
                            }
                    }
            goto next;
found_free_mach:
            if(!no_free_mach){
                    if(temp_m->mstate != UN_UTILIZED){
                            for(temp_pi = temp_d->part_info; temp_pi != NULL &&\
                                    !(temp_pi->ptype == temp_m->setup_for) &&\
```

```
                                        (temp_pi->repeat == temp_m->repeat));\
                                                temp_pi = temp_pi->next);
                        if(temp_pi == NULL) print_errorp();
                        n_pre = nmach_proc_ptype(temp_d, temp_pi->repeat,\
                                temp_pi->ptype);
                        if(temp_pi->pre >= 1){
                                schedule(LOAD, time_now+1L, 0,\
                                temp_d->dept_no, temp_m->repeat, temp_m->setup_for);
                        }
                }
        }
    next:
        ;
        }
    }
}

int nmach_proc_ptype(dept_ptr, repeat, ptype)
DEPT *dept_ptr;
int repeat, ptype;
{
    MACHINE *mach_ptr;
    int count = 0;

    for(mach_ptr = dept_ptr->mach; mach_ptr != NULL;\
                        mach_ptr = mach_ptr->next){
        if((mach_ptr->repeat == repeat) && (mach_ptr->setup_for == ptype)){
            if(mach_ptr->mstate == 2){
                count++;
            }
        }
    }
    return(count);
}


int update_zoom_window(mach_no, mstate, repeat, part_type)
int mach_no, mstate, repeat;
char part_type;
{
    int row, col;

    row = ((zoom_row+1)+((mach_no-1)*2));
    col = zoom_col+1;
    switch(mstate){
        case 0:
            viprintf(1, row, col, MACHINE_COLOR, "%c", IDLE);
            break;
        case 1:
            if(repeat > 0){
                viprintf(1, row, col-3, B_BLUE+F_GREEN+INT, "%1d", repeat);
```

```
                }else{
                        viprintf(1, row, col-3, B_BLUE+F_GREEN+INT, "%c", ' ');
                }
                viprintf(1, row, col-2, B_BLUE+F_GREEN+INT, "%c", part_type);
                viprintf(1, row, col, MACHINE_COLOR+BLNK, "%c", SETTING_UP);
                break;
        case 2:
                viprintf(1, row, col, MACHINE_COLOR+BLNK, "%c", PROCESSING);
                break;
        default:
                break;
    }
}

/* Function to update the inventory costs in statistics window */
static void update_inv_finance()
{
    int i, j, row, col;
    long inv_costs;
    RAW *temp_raw;
    PRODUCT *temp_prod;
    int raw_cost[10], prod_cost[10], no_of_opns[10];

    inv_costs = 0L;
    for(i=0, temp_raw = raw_mat; temp_raw != NULL; temp_raw = temp_raw->next, i++){
        raw_cost[i] = temp_raw->cost;
    }
    for(i=0, temp_prod = product; temp_prod != NULL; temp_prod = temp_prod->next, i++){
        prod_cost[i] = temp_prod->price;
    }
    for(i = 0; i < n_ptypes; i++){
        for(j = 0; route[i][j] != 0; j++){
            ;
        }
        no_of_opns[i] = j;
    }
    for(i = 0; i < n_ptypes; i++){
        for(j = 0; route[i][j] != 0; j++){
            inv_costs = inv_costs + (count[i][j] * (raw_cost[i] + j * \
                    ((prod_cost[i] - raw_cost[i])/no_of_opns[i])))/10;
        }
    }
    row = stats->fin_row;
    col = stats->fin_col;
    viprintf(3, row+5, col, STAT_FIN_COLOR, "%7ld", inv_costs);
    expense=inv_costs+stats->raw_mat_exp+stats->mat_hand_exp+stats->other_op_exp;
    cash = stats->start_cash + stats->sales_revenue - expense;
    stats->cash_now = cash;
    update_finance();
}
```

```
void init_on_hand()
{
     int i, j, dept_no;
     DEPT *temp_d;
     PART_INFO *temp_pi;

     for(i = 0; i < n_ptypes; i++){
            for(j = 0; route[i][j] != 0; j++){
                   dept_no = route[i][j];
                   for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no;\
                                        temp_d = temp_d->next);
                   for(temp_pi = temp_d->part_info;\
                                 temp_pi != NULL; temp_pi = temp_pi->next){
                          if(route[i][j+1] == 0){
                                 update_on_hand(temp_pi);
                          }
                   }
            }
     }
}

static void update_on_hand(ptr_pi)
PART_INFO *ptr_pi;
{
     int row = 16;
     int col = 23;
     int dx = 0;

     dx = ptr_pi->ptype;
     viprintf(3, row+dx-1, col, STAT_SLS_COLOR, "%3d", ptr_pi->post);
}

/* Function to update inventory information on the statistics window */
void update_stat_inv_info()
{
     int row = 16;
     int col = 28;
     DEPT *temp_d;
     PART_INFO *temp_pi, *temp_dummy_pi;
     int i, j, k, dept_no, repeat;
     int inventory[10];

     for(i = 0; i < n_ptypes; i++){
            for(j = 0; route[i][j] != 0; j++){
                   count[i][j] = 0;
            }
            inventory[i] = 0;
     }
     for(i = 0; i < n_ptypes; i++){
            k = 0;
            for(j = 0; route[i][j] != 0; j++){
```

```
                repeat = FALSE;
                dept_no = route[i][j];
                for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no;\
                                temp_d = temp_d->next);
                for(temp_pi = temp_d->part_info;\
                        temp_pi != NULL && temp_pi->ptype != (i+1);\
                                temp_pi = temp_pi->next);
                for(temp_dummy_pi = temp_pi; temp_dummy_pi != NULL && \
                        temp_dummy_pi->ptype == (i+1);\
                                temp_dummy_pi = temp_dummy_pi->next){
                        if(temp_dummy_pi->repeat == 1){
                                k = k +1;
                                repeat = TRUE;
                                if(k != 1){
                                        if(route[i][j+1] == 0){
                                        count[i][j] = temp_dummy_pi->pre + count[i][j];
                                        }else{
                                        count[i][j]=temp_dummy_pi->pre+temp_dummy_pi->post\
                                                + count[i][j];
                                        }
                                }
                        }else{
                                repeat = FALSE;
                        }
                }
                if(repeat == FALSE){
                        if(route[i][j+1] == 0){
                        count[i][j] = temp_pi->pre + count[i][j];
                        }else{
                        count[i][j] = temp_pi->pre + temp_pi->post + count[i][j];
                        }
                }else{
                        if(k == 1){
                                if(route[i][j+1] == 0){
                                count[i][j] = temp_pi->pre + count[i][j];
                                }else{
                                count[i][j]=temp_pi->pre+temp_pi->post+count[i][j];
                                }
                        }
                }
        }
    }
    for(i = 0; i < n_ptypes; i++, row++){
            for(j = 0; route[i][j] != 0; j++){
                    inventory[i] = inventory[i] + count[i][j];
            }
            viprintf(3, row, col, STAT_SLS_COLOR, "%6d", inventory[i]);
    }

    update_inv_finance();
}
```

```
#include <stdio.h>
#include "user\opt.h"
#include "user\vi.h"
#include "user\std.h"

/* Functions defined in setscr.c --- */
void set_screen(void);
void draw_depts(void);
void draw_routing(void);
void draw_raw_mats(void);
void draw_products(void);
void draw_menu(int);
void print_modes(void);
void reset_quantity(void);
void draw_initial_scr(void);

/* Function to paint the initial screen */
void set_screen()
{
    int i, row, col;

    vicls(0);
    viprintf(0, 0, 0, S_TEXT_COLOR, "%s", "WEEK  DAY");
    viprintf(0, 1, 1, CLOCK_COLOR, "%s", "00");
    viprintf(0, 1, 7, CLOCK_COLOR, "%s", "0");
    viprintf(0, 3, 2, S_TEXT_COLOR, "%s", "TIME:");
    viprintf(0, 4, 2, CLOCK_COLOR, "%s", "00:00");
    viprintf(0, 6, 0, S_TEXT_COLOR, "%s", "CASH");
    viprintf(0, 7, 0, CASH_COLOR, "%s", "       ");
    viprintf(0, 9, 0, S_TEXT_COLOR, "%s", "EXPENSE");
    viprintf(0,10, 0, EXPENSE_COLOR, "%s", "       ");
    viprintf(0,12, 0, S_TEXT_COLOR, "%s", "Pace:");
    viprintf(0,12, 6, CLOCK_COLOR, "%s", "1");

    for(row=0; row<22; row++){
        viprintf(0, row, 9, TITLE_COLOR, "%c", '\xB3');
    }
    viprintf(0, 5, 9, TITLE_COLOR, "%c", '\xB4');
    viprintf(0,11, 9, TITLE_COLOR, "%c", '\xB4');
    viprintf(0,13, 9, TITLE_COLOR, "%c", '\xB4');
    viprintf(0,21, 9, TITLE_COLOR, "%c", '\xD9');
    for(col=0; col<9; col++){
        viprintf(0, 5, col, TITLE_COLOR, "%c", '\xC4');
        viprintf(0,11, col, TITLE_COLOR, "%c", '\xC4');
        viprintf(0,13, col, TITLE_COLOR, "%c", '\xC4');
        viprintf(0,21, col, TITLE_COLOR, "%c", '\xC4');
    }
    if(state == TRUE){
        viprintf(0, 1, 1, CLOCK_COLOR, "%2d", week_now);
        if(week_now < 10) viprintf(0, 1, 1, CLOCK_COLOR, "%c", '0');
        viprintf(0, 1, 7, CLOCK_COLOR, "%1d", day_now);
```

```
            viprintf(0, 4, 2, CLOCK_COLOR, "%2d:%2d", hr_now, min_now);
            if(min_now < 10) viprintf(0, 4, 5, CLOCK_COLOR, "%c", '0');
      }
      draw_menu(1);
      draw_menu(2);
      draw_menu(3);
      draw_menu(4);
      draw_menu(5);
      draw_depts();
      if(state == TRUE){
            draw_initial_scr();
      }
      draw_routing();
      draw_raw_mats();
      draw_products();
      update_finance();
      position_cursor(0, 23, 0);
      position_cursor(1, 23, 0);
      position_cursor(2, 23, 0);
      position_cursor(3, 23, 0);
}

/* Function to draw the menus */
void draw_menu(menu_no)
int menu_no;
{
      int i;

      if(menu_no == 1){
            for(i=0; i < 80; i++){
                  viprintf(0,24, i, MENU_KEY_COLOR, "%c", ' ');
            }
            viprintf(0,24, 0, MENU_KEY_COLOR, "%s", "F2");
            viprintf(0,24, 2, MENU_FNC_COLOR, "%s", "Zoom");
            viprintf(0,24, 8, MENU_KEY_COLOR, "%s", "F3");
            viprintf(0,24,10, MENU_FNC_COLOR, "%s", "Calendar");
            viprintf(0,24,20, MENU_KEY_COLOR, "%s", "F4");
            viprintf(0,24,22, MENU_FNC_COLOR, "%s", "Stat");
            viprintf(0,24,28, MENU_KEY_COLOR, "%s", "F");
            viprintf(0,24,29, MENU_FNC_COLOR, "%s", "reeze");
            viprintf(0,24,36, MENU_KEY_COLOR, "%s", "B");
            viprintf(0,24,37, MENU_FNC_COLOR, "%s", "reakpoint");
            viprintf(0,24,48, MENU_KEY_COLOR, "%s", "S");
            viprintf(0,24,49, MENU_FNC_COLOR, "%s", "chedule");
            viprintf(0,24,58, MENU_KEY_COLOR, "%s", "P");
            viprintf(0,24,59, MENU_FNC_COLOR, "%s", "rint");
            viprintf(0,24,65, MENU_KEY_COLOR, "%s", "Q");
            viprintf(0,24,66, MENU_FNC_COLOR, "%s", "uit");
            viprintf(0,24,71, MENU_KEY_COLOR, "%s", "<+/->");
            viprintf(0,24,76, MENU_FNC_COLOR, "%s", "Pace");
      }
```

```
if(menu_no == 2){
        for(i=0; i < 80; i++){
                viprintf(1,24, i, MENU_KEY_COLOR, "%c", ' ');
        }
        viprintf(1,24, 0, MENU_KEY_COLOR, "%s", "F1");
        viprintf(1,24, 2, MENU_FNC_COLOR, "%s", "Overview");
        viprintf(1,24,12, MENU_KEY_COLOR, "%s", "F2");
        viprintf(1,24,14, MENU_FNC_COLOR, "%s", "Dept#");
        viprintf(1,24,21, MENU_KEY_COLOR, "%s", "F3");
        viprintf(1,24,23, MENU_FNC_COLOR, "%s", "Calendar");
        viprintf(1,24,33, MENU_KEY_COLOR, "%s", "F4");
        viprintf(1,24,35, MENU_FNC_COLOR, "%s", "Stat");
        viprintf(1,24,41, MENU_KEY_COLOR, "%s", "F");
        viprintf(1,24,42, MENU_FNC_COLOR, "%s", "reeze");
        viprintf(1,24,49, MENU_KEY_COLOR, "%s", "C");
        viprintf(1,24,50, MENU_FNC_COLOR, "%s", "hange");
        viprintf(1,24,57, MENU_KEY_COLOR, "%s", "P");
        viprintf(1,24,58, MENU_FNC_COLOR, "%s", "rint");
        viprintf(1,24,64, MENU_KEY_COLOR, "%s", "Q");
        viprintf(1,24,65, MENU_FNC_COLOR, "%s", "uit");
        viprintf(1,24,70, MENU_KEY_COLOR, "%s", "<+/->");
        viprintf(1,24,75, MENU_FNC_COLOR, "%s", "Pace");
}
if(menu_no == 3){
        for(i=0; i < 80; i++){
                viprintf(2,24, i, MENU_KEY_COLOR, "%c", ' ');
        }
        viprintf(2,24, 0, B_BLUE+F_BROWN+INT, "%s", "ScanKeys");
        viprintf(2,24, 9, MENU_KEY_COLOR, "%s %c%c", "PgUp PgDn", '\x18', '\x19');
        viprintf(2,24,23, MENU_KEY_COLOR, "%s", "F1");
        viprintf(2,24,25, MENU_FNC_COLOR, "%s", "Overview");
        viprintf(2,24,35, MENU_KEY_COLOR, "%s", "F2");
        viprintf(2,24,37, MENU_FNC_COLOR, "%s", "Zoom");
        viprintf(2,24,43, MENU_KEY_COLOR, "%s", "F4");
        viprintf(2,24,45, MENU_FNC_COLOR, "%s", "Stat");
        viprintf(2,24,51, MENU_KEY_COLOR, "%s", "E");
        viprintf(2,24,52, MENU_FNC_COLOR, "%s", "dit");
        viprintf(2,24,57, MENU_KEY_COLOR, "%s", "F");
        viprintf(2,24,58, MENU_FNC_COLOR, "%s", "reeze");
        viprintf(2,24,65, MENU_KEY_COLOR, "%s", "Q");
        viprintf(2,24,66, MENU_FNC_COLOR, "%s", "uit");
        viprintf(2,24,71, MENU_KEY_COLOR, "%s", "<+/->");
        viprintf(2,24,76, MENU_FNC_COLOR, "%s", "Pace");
}
if(menu_no == 4){
        for(i=0; i < 72; i++){
                viprintf(3,24, i, MENU_KEY_COLOR, "%c", ' ');
        }
        viprintf(3,24, 9, MENU_KEY_COLOR, "%s", "F1");
        viprintf(3,24,11, MENU_FNC_COLOR, "%s", "Overview");
        viprintf(3,24,21, MENU_KEY_COLOR, "%s", "F2");
```

```
        viprintf(3,24,23, MENU_FNC_COLOR, "%s", "Zoom");
        viprintf(3,24,29, MENU_KEY_COLOR, "%s", "F3");
        viprintf(3,24,31, MENU_FNC_COLOR, "%s", "Calendar");
        viprintf(3,24,41, MENU_KEY_COLOR, "%s", "F");
        viprintf(3,24,42, MENU_FNC_COLOR, "%s", "reeze");
        viprintf(3,24,49, MENU_KEY_COLOR, "%s", "P");
        viprintf(3,24,50, MENU_FNC_COLOR, "%s", "rint");
        viprintf(3,24,56, MENU_KEY_COLOR, "%s", "Q");
        viprintf(3,24,57, MENU_FNC_COLOR, "%s", "uit");
        viprintf(3,24,62, MENU_KEY_COLOR, "%s", "<+/->");
        viprintf(3,24,67, MENU_FNC_COLOR, "%s", "Pace");
}
if(menu_no == 5){
        viprintf(0,24,28, MENU_KEY_COLOR, "%s", "R");
        viprintf(0,24,29, MENU_FNC_COLOR, "%s", "esume");
        viprintf(1,24,41, MENU_KEY_COLOR, "%s", "R");
        viprintf(1,24,42, MENU_FNC_COLOR, "%s", "esume");
        viprintf(2,24,57, MENU_KEY_COLOR, "%s", "R");
        viprintf(2,24,58, MENU_FNC_COLOR, "%s", "esume");
        viprintf(3,24,41, MENU_KEY_COLOR, "%s", "R");
        viprintf(3,24,42, MENU_FNC_COLOR, "%s", "esume");
}
if(menu_no == 6){
        viprintf(0,24,28, MENU_KEY_COLOR, "%s", "F");
        viprintf(0,24,29, MENU_FNC_COLOR, "%s", "reeze");
        viprintf(1,24,41, MENU_KEY_COLOR, "%s", "F");
        viprintf(1,24,42, MENU_FNC_COLOR, "%s", "reeze");
        viprintf(2,24,57, MENU_KEY_COLOR, "%s", "F");
        viprintf(2,24,58, MENU_FNC_COLOR, "%s", "reeze");
        viprintf(3,24,41, MENU_KEY_COLOR, "%s", "F");
        viprintf(3,24,42, MENU_FNC_COLOR, "%s", "reeze");
}
if(menu_no == 7){
        for(i=22; i < 80; i++){
                viprintf(2, 24, i, MENU_KEY_COLOR, "%c", ' ');
        }
        viprintf(2, 24, 36, MENU_KEY_COLOR, "%s", "O");
        viprintf(2, 24, 37, MENU_FNC_COLOR, "%s", "eleteEvent");
        viprintf(2, 24, 49, MENU_KEY_COLOR, "%s", "A");
        viprintf(2, 24, 50, MENU_FNC_COLOR, "%s", "ddEvent");
        viprintf(2, 24, 59, MENU_KEY_COLOR, "%s", "Esc");
        viprintf(2, 24, 62, MENU_FNC_COLOR, "%s", "Oone");
}
if(menu_no == 8){
        for(i=22; i < 80; i++){
                viprintf(2, 24, i, MENU_KEY_COLOR, "%c", ' ');
        }
        viprintf(2, 24, 22, B_BLUE+F_RED+INT, "%s", "Select Event by ScanKeys");
        viprintf(2, 24, 48, MENU_KEY_COLOR, "%s", "Del");
        viprintf(2, 24, 51, MENU_FNC_COLOR, "%s", "Delete");
        viprintf(2, 24, 59, MENU_KEY_COLOR, "%s", "Ins");
```

```
        viprintf(2, 24, 62, MENU_FNC_COLOR, "%s", "Undelete");
        viprintf(2, 24, 72, MENU_KEY_COLOR, "%s", "Esc");
        viprintf(2, 24, 75, MENU_FNC_COLOR, "%s", "Done");
}
if(menu_no == 9){
        for(i=22; i < 80; i++){
                viprintf(2, 24, i, MENU_KEY_COLOR, "%c", ' ');
        }
        viprintf(2,24,23, MENU_KEY_COLOR, "%s", "F1");
        viprintf(2,24,25, MENU_FNC_COLOR, "%s", "Overview");
        viprintf(2,24,35, MENU_KEY_COLOR, "%s", "F2");
        viprintf(2,24,37, MENU_FNC_COLOR, "%s", "Zoom");
        viprintf(2,24,43, MENU_KEY_COLOR, "%s", "F4");
        viprintf(2,24,45, MENU_FNC_COLOR, "%s", "Stat");
        viprintf(2,24,51, MENU_KEY_COLOR, "%s", "E");
        viprintf(2,24,52, MENU_FNC_COLOR, "%s", "dit");
        if(!freezed){
                viprintf(2,24,57, MENU_KEY_COLOR, "%s", "F");
                viprintf(2,24,58, MENU_FNC_COLOR, "%s", "reeze");
        }else{
                viprintf(2,24,57, MENU_KEY_COLOR, "%s", "R");
                viprintf(2,24,58, MENU_FNC_COLOR, "%s", "esume");
        }
        viprintf(2,24,65, MENU_KEY_COLOR, "%s", "Q");
        viprintf(2,24,66, MENU_FNC_COLOR, "%s", "uit");
        viprintf(2,24,71, MENU_KEY_COLOR, "%s", "(+/-)");
        viprintf(2,24,76, MENU_FNC_COLOR, "%s", "Pace");
}
if(menu_no == 10){
        if(pageno == 0){
                for(i=0; i < 80; i++){
                        viprintf(0,24, i, MENU_KEY_COLOR, "%c", ' ');
                }
                viprintf(0, 24, 15, MENU_KEY_COLOR, "%s", "W");
                viprintf(0, 24, 16, MENU_FNC_COLOR, "%s", "rite");
                viprintf(0, 24, 22, MENU_KEY_COLOR, "%s", "P");
                viprintf(0, 24, 23, MENU_FNC_COLOR, "%s", "urchase");
                viprintf(0, 24, 32, MENU_KEY_COLOR, "%s", "S");
                viprintf(0, 24, 33, MENU_FNC_COLOR, "%s", "etup");
                viprintf(0, 24, 39, MENU_KEY_COLOR, "%s", "L");
                viprintf(0, 24, 40, MENU_FNC_COLOR, "%s", "oad");
                viprintf(0, 24, 45, MENU_KEY_COLOR, "%s", "M");
                viprintf(0, 24, 46, MENU_FNC_COLOR, "%s", "ove");
                viprintf(0, 24, 51, MENU_KEY_COLOR, "%s", "Esc");
                viprintf(0, 24, 54, MENU_FNC_COLOR, "%s", "Done");
        }
        if(pageno == 2){
                for(i=22; i < 80; i++){
                        viprintf(2,24, i, MENU_KEY_COLOR, "%c", ' ');
                }
                viprintf(2, 24, 33, MENU_KEY_COLOR, "%s", "P");
```

```
                viprintf(2, 24, 34, MENU_FNC_COLOR, "%s", "urchase");
                viprintf(2, 24, 43, MENU_KEY_COLOR, "%s", "S");
                viprintf(2, 24, 44, MENU_FNC_COLOR, "%s", "etup");
                viprintf(2, 24, 50, MENU_KEY_COLOR, "%s", "L");
                viprintf(2, 24, 51, MENU_FNC_COLOR, "%s", "oad");
                viprintf(2, 24, 56, MENU_KEY_COLOR, "%s", "M");
                viprintf(2, 24, 57, MENU_FNC_COLOR, "%s", "ove");
                viprintf(2, 24, 62, MENU_KEY_COLOR, "%s", "Esc");
                viprintf(2, 24, 65, MENU_FNC_COLOR, "%s", "Done");
        }
    }
    if(menu_no == 11){
        for(i=0; i < 80; i++){
            viprintf(0,24, i, MENU_KEY_COLOR, "%c", ' ');
        }
        viprintf(0,24, 0, MENU_KEY_COLOR, "%s", "F2");
        viprintf(0,24, 2, MENU_FNC_COLOR, "%s", "Zoom");
        viprintf(0,24, 8, MENU_KEY_COLOR, "%s", "F3");
        viprintf(0,24,10, MENU_FNC_COLOR, "%s", "Calendar");
        viprintf(0,24,20, MENU_KEY_COLOR, "%s", "F4");
        viprintf(0,24,22, MENU_FNC_COLOR, "%s", "Stat");
        if(!freezed){
            viprintf(0,24,28, MENU_KEY_COLOR, "%s", "F");
            viprintf(0,24,29, MENU_FNC_COLOR, "%s", "reeze");
        }else{
            viprintf(0,24,28, MENU_KEY_COLOR, "%s", "R");
            viprintf(0,24,29, MENU_FNC_COLOR, "%s", "esume");
        }
        viprintf(0,24,36, MENU_KEY_COLOR, "%s", "B");
        viprintf(0,24,37, MENU_FNC_COLOR, "%s", "reakpoint");
        viprintf(0,24,48, MENU_KEY_COLOR, "%s", "S");
        viprintf(0,24,49, MENU_FNC_COLOR, "%s", "chedule");
        viprintf(0,24,58, MENU_KEY_COLOR, "%s", "P");
        viprintf(0,24,59, MENU_FNC_COLOR, "%s", "rint");
        viprintf(0,24,65, MENU_KEY_COLOR, "%s", "Q");
        viprintf(0,24,66, MENU_FNC_COLOR, "%s", "uit");
        viprintf(0,24,71, MENU_KEY_COLOR, "%s", "<+/->");
        viprintf(0,24,76, MENU_FNC_COLOR, "%s", "Pace");
    }
}

/* Function to draw the departments */
void draw_depts()
{
    int i, j, k, row, col;
    DEPT *temp_d;
    MACHINE *temp_m;
    PART_INFO *temp_pi;

    for(temp_d = dept_h; temp_d != NULL; temp_d = temp_d->next){
        viprintf(0, temp_d->row-1, temp_d->col-5, TITLE_COLOR, "%s %d", \
```

```
                "Department", temp_d->dept_no);
        for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
                row = temp_m->row;
                col = temp_m->col;
                viprintf(0, row, col, B_RED+F_WHITE+INT, "%d", temp_m->mach_no);
                viprintf(0, row+1, col-2, B_BLUE+F_BROWN+INT, "%c", '\xB3');
                viprintf(0, row+1, col+2, B_BLUE+F_BROWN+INT, "%c", '\xB3');
                viprintf(0, row+1, col-1, B_BLUE+F_WHITE+INT, "%s", "   ");
        }
        for(temp_pi = temp_d->part_info; temp_pi != NULL; temp_pi = temp_pi->next){
                row = temp_pi->row;
                col = temp_pi->col;
                if(temp_pi->repeat > 0){
                viprintf(0, row, col-2, B_BLACK+F_GREEN+INT, "%d", temp_pi->repeat);
                viprintf(0, row, col+19, B_BLACK+F_GREEN+INT, "%d", temp_pi->repeat);
                }
                viprintf(0,row,col-1,B_BLACK+F_BROWN+INT,"%c",temp_pi->ptype_in_char);
                viprintf(0,row,col+20,B_BLACK+F_BROWN+INT,"%c",temp_pi->ptype_in_char);
                if(temp_pi->pre > 0){
                        viprintf(0, row, col, PRE_COLOR, "%3d", temp_pi->pre);
                }else{
                        viprintf(0, row, col, PRE_COLOR, "%s", "   ");
                }
                if(temp_pi->post > 0){
                        viprintf(0, row, col+16, POST_COLOR, "%3d", temp_pi->post);
                }else{
                        viprintf(0, row, col+16, POST_COLOR, "%s", "   ");
                }
                viprintf(0, row, col+3, B_GREEN+F_BROWN+INT, "%c", '\xB3');
                viprintf(0, row, col+15, B_GREEN+F_BROWN+INT, "%c", '\xB3');
                viprintf(0, row, col+4, TIME_COLOR, "%s", "   ");
                viprintf(0, row, col+12, MTIME_COLOR, "%s", "   ");
        }
    }
}

/* Function to print the routing matrix */
void draw_routing()
{
    int i, j, row, col;
    char pchar;

    row = routing_row;
    col = routing_col;

    viprintf(0, row, col-1, TITLE_COLOR, "%s", "ROUTING");
    viprintf(0, ++row, col-5, ROUTE_COLOR, "%s", "Type    route");
    for(i=0; i < n_ptypes; i++){
            col = routing_col;
            map_part_type(i+1, &pchar);
            viprintf(0, ++row, col-4, ROUTE_COLOR, "%c %c", pchar, ':');
```

```
            for(j=0; j < max_ops; j++){
                    if(route[i][j] > 0){
                            viprintf(0, row, col++, ROUTE_COLOR, "%d", route[i][j]);
                            if(route[i][j+1] > 0){
                                    viprintf(0, row, col++, ROUTE_COLOR, "%c", '\xAF');
                            }
                    }
            }
    }
}

/* Function to draw the raw materials and print the costs */
void draw_raw_mats()
{
    int row, col;
    RAW *temp_raw;
    char pchar;

    row = raw_mat_row;
    col = raw_mat_col;
    viprintf(0, row, col, TITLE_COLOR, "%s", "Raw Materials:");
    for(temp_raw = raw_mat; temp_raw != NULL; temp_raw = temp_raw->next){
            map_part_type(temp_raw->raw_mat_no, &pchar);
            viprintf(0, temp_raw->row, temp_raw->col, CAL_FNC_COLOR,\
                            " %c: ", pchar);
            viprintf(0, temp_raw->row, temp_raw->col+4, REV,\
                            " %c%2d ", '$', temp_raw->cost);
    }
}

/* Function to draw the products */
void draw_products()
{
    int row, col;
    PRODUCT *temp_prod;
    char pchar;

    row = prod_row;
    col = prod_col;
    viprintf(0, row, col, TITLE_COLOR, "%s", "Products to be sold:");
    for(temp_prod = product; temp_prod != NULL; temp_prod = temp_prod->next){
            map_part_type(temp_prod->prod_no, &pchar);
            viprintf(0, temp_prod->row, temp_prod->col,\
                            PROD_NO_COLOR, "%c:", pchar);
            viprintf(0, temp_prod->row, temp_prod->col+2,\
             PRODUCT_COLOR, "%5d", temp_prod->limit-temp_prod->qnty_produced);
    }
}

/* Function to reset the quantity produced for the next week */
void reset_quantity()
```

```
{
     PRODUCT *temp_prod;

     for(temp_prod = product; temp_prod != NULL; temp_prod = temp_prod->next){
          if(temp_prod->limit-temp_prod->qnty_produced == 0){
               temp_prod->qnty_produced = 0;
          }
     }
}

/* Function to print modes */
void print_modes()
{
     #define MODE_TXT_COLOR  B_BLACK+F_BROWN+INT
     #define MODE_COLOR      B_BLUE+F_CYAN+INT

     viprintf(0, 14, 2, TITLE_COLOR, "%s", "MODES");
     viprintf(0, 15, 0, MODE_TXT_COLOR, "%s", "Purchase:");
     viprintf(0, 17, 0, MODE_TXT_COLOR, "%s", "Setup:");
     viprintf(0, 19, 0, MODE_TXT_COLOR, "%s", "Move:");
     if(auto_purchase){
          viprintf(0, 16, 3, MODE_COLOR, "%s", " Auto ");
     }else{
          viprintf(0, 16, 3, MODE_COLOR, "%s", "Manual");
     }
     if(auto_setup){
          viprintf(0, 18, 3, MODE_COLOR, "%s", " Auto ");
     }else{
          viprintf(0, 18, 3, MODE_COLOR, "%s", "Manual");
     }
     if(auto_move){
          viprintf(0, 20, 3, MODE_COLOR, "%s", " Auto ");
     }else{
          viprintf(0, 20, 3, MODE_COLOR, "%s", "Manual");
     }
}

void draw_initial_scr()
{
     int row, col;
     DEPT *temp_d;
     MACHINE *temp_m;

     for(temp_d = dept_h; temp_d != NULL; temp_d = temp_d->next){
          for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
               row = temp_m->row;
               col = temp_m->col;
               switch (temp_m->mstate){
               case 0:
                    if(temp_m->setup_for != UN_UTILIZED){
                         if(temp_m->repeat > 0){
```

```
                                viprintf(0, row+1, col-1, B_BLUE+F_GREEN+INT,\
                                        "%1d", temp_m->repeat);
                        }
                viprintf(0,row+1,col,MACHINE_COLOR,"%c",temp_m->setup_for_in_char);
                viprintf(0, row+1, col+1, MACHINE_COLOR, "%c", IDLE);
                }
                break;
        case 1:
                if(temp_m->repeat > 0){
                        viprintf(0, row+1, col-1, B_BLUE+F_GREEN+INT, \
                                "%1d", temp_m->repeat);
                }
                viprintf(0, row+1, col, MACHINE_COLOR, "%c", \
                        temp_m->setup_for_in_char);
                viprintf(0,row+1,col+1,MACHINE_COLOR,"%c",SETTING_UP);
                break;
        case 2:
                if(temp_m->repeat > 0){
                viprintf(0, row+1, col-1, B_BLUE+F_GREEN+INT, "%1d", temp_m->repeat);
                }
                viprintf(0, row+1, col, MACHINE_COLOR, "%c", \
                        temp_m->setup_for_in_char);
                viprintf(0,row+1,col+1,MACHINE_COLOR,"%c",PROCESSING);
                break;
        }
    }
}
```

```
#include <stdio.h>
#include "user\opt.h"
#include "user\vi.h"
#include "user\std.h"

/* Function to initialize the statistics structure */
int init_stat()
{
     if(state == FALSE){
            stats = (STATS*)my_malloc(sizeof(STATS));
            stats->fin_row = 5;
            stats->fin_col = 27;
            stats->start_cash = cash;
            stats->cash_now = cash;
            stats->sales_revenue = 0L;
            stats->raw_mat_exp = 0;
            stats->mat_hand_exp = 0;
            stats->other_op_exp = 0;
            stats->net_cash_flow = 0;
     }
     draw_stat_screen();
}

/* Function to paint the statistics window */
int draw_stat_screen()
{
     int i, row, col;
     char pchar;
     DEPT *temp_d;
     MACHINE *temp_m;

     vicls(3);
     viprintf(3, 0, 33, TITLE_COLOR, "%s", "STATISTICS");
     viprintf(3, 0, 5, S_TEXT_COLOR, "%s", "WEEK:     DAY:");
     viprintf(3, 0, 63, S_TEXT_COLOR, "%s", "TIME:");
     for(row=2; row < 14; row++){
            for(col=5; col < 36; col++){
                   viputc(3, row, col, STAT_FIN_COLOR, ' ');
            }
     }
     for(row=14; row < n_ptypes+3+14; row++){
            for(col=5; col < 36; col++){
                   viputc(3, row, col, STAT_SLS_COLOR, ' ');
            }
     }
     for(row=2; row < n_ptypes+3+14; row++){
            for(col=36; col < 74; col++){
                   viputc(3, row, col, STAT_ACT_COLOR, ' ');
            }
     }
     viprintf(3, 2,12, STAT_FIN_COLOR-F_WHITE+F_BROWN, "%s", "FINANCIAL REPORT");
```

```
    viprintf(3, 4,13, STAT_FIN_COLOR, "%s", "Starting Cash");
    viprintf(3, 5,18, STAT_FIN_COLOR, "%s", "Cash Now");
    viprintf(3, 7,10, STAT_FIN_COLOR, "%s", "Raw Mat Expenses");
    viprintf(3, 8, 5, STAT_FIN_COLOR, "%s", "Mat Handling Expenses");
    viprintf(3, 9, 9, STAT_FIN_COLOR, "%s", "Other Op Expenses");
    viprintf(3,10, 5, STAT_FIN_COLOR, "%s", "Work_in_process Value");
    viprintf(3,11, 8, STAT_FIN_COLOR, "%s", "Revenue from Sales");
    viprintf(3,12,13, STAT_FIN_COLOR, "%s", "Net Cash Flow");
    viprintf(3,14,14, STAT_SLS_COLOR-F_WHITE+F_BROWN, "%s", "SALES REPORT");
    viprintf(3,15,16, STAT_SLS_COLOR, "%s", "Sold");
    viprintf(3,15,22, STAT_SLS_COLOR, "%s", "On Hand");
    viprintf(3,15,31, STAT_SLS_COLOR, "%s", "WIP");

    viprintf(3, 2,47, STAT_ACT_COLOR-F_WHITE+F_BROWN, "%s", "ACTIVATION REPORT");
    viprintf(3, 4,46, STAT_ACT_COLOR, "%s", "Mach#  Prod%  Setup%  Idle%");
    for(i=0, row=16; i < n_ptypes; i++, row++){
            map_part_type(i+1, &pchar);
            viprintf(3, row, 5, STAT_SLS_COLOR, "Product %c:", pchar);
    }
    for(temp_d=dept_h, row=5; temp_d != NULL; temp_d = temp_d->next){
            viprintf(3, row, 37, STAT_ACT_COLOR, "Dept %1d", temp_d->dept_no);
            for(temp_m=temp_d->mach;temp_m!=NULL;temp_m=temp_m->next,row++){
                    viprintf(3,row,48,STAT_ACT_COLOR,"%1d",temp_m->mach_no);
            }
    }
}


int calc_cash_flow()
{
    cash -= OTHER_OP_EXPENSE;
    expense += OTHER_OP_EXPENSE;
    update_finance();
    stats->other_op_exp += OTHER_OP_EXPENSE;
    stats->cash_now -= OTHER_OP_EXPENSE;
    stats->net_cash_flow = stats->cash_now - stats->start_cash;
    if(pageno != 3){
            update_stat_clock();
            update_stat_prod_info();
            update_act_report();
            pageno = 3;
            set_page(pageno);
    }
    update_stat_finance();
    viprintf(3, 22, 6, MSG_COLOR, "This is end of week %d statistics. \
             'P' to print/'R' to continue", week_now-1);
}

int reset_finance()
{
    cash = stats->cash_now;
```

```
    expense = 0L;
    stats->start_cash = stats->cash_now;
    stats->cash_now = cash;
    stats->sales_revenue = 0L;
    stats->raw_mat_exp = 0;
    stats->mat_hand_exp = 0;
    stats->other_op_exp = 0;
    stats->net_cash_flow = 0;
    viblank(3, 22, 0, 22, 79);
    update_finance();
}
```

```
#include <stdio.h>
#include "user\opt.h"
#include "user\calendar.h"

/* Function to breakup time into weeks, days, hours and minutes */
int time_to_string(time_in_mins, p_cal)
unsigned long time_in_mins;
CALENDAR *p_cal;
{
    p_cal->weeks = (int)(time_in_mins/MIN_PER_WEEK);
    p_cal->days = (int)((time_in_mins/MIN_PER_DAY)-\
                  ((unsigned long)(p_cal->weeks)*DAYS_PER_WEEK));
    p_cal->hrs = (int)((time_in_mins/MIN_PER_HR)-\
                 ((unsigned long)(p_cal->weeks)*HRS_PER_WEEK)-\
                 ((unsigned long)(p_cal->days)*HRS_PER_DAY));
    p_cal->mins = (int)(time_in_mins % MIN_PER_HR);
}

/* to be changed to structure version */
int string_to_time(time_in_mins, weeks, days, hrs, mins)
unsigned long *time_in_mins;
int weeks, days, hrs, mins;
{
    *time_in_mins = (unsigned long)weeks*MIN_PER_WEEK +
                 (unsigned long)days*MIN_PER_DAY +
                 (unsigned long)hrs*MIN_PER_HR +
                 (unsigned long)mins;
    /* printf("time_in_mins = %lu\n", *time_in_mins); */
}
```

```
#include <stdio.h>
#include <dos.h>
#include <graph.h>
#include <signal.h>
#include <stdlib.h>
#include <process.h>
#include "user\std.h"
#include "user\doslib.h"
#include "user\keydefs.h"
#include "user\bioslib.h"
#include "user\opt.h"
#include "user\vi.h"

/* --- Functions defined in Utils.c --- */
int getkey(void);
unsigned int position_cursor(int, int, int);
void set_page(int);
void hide_cursor(void);
int map_part_type(int, char*);
int map_char(char);
void beep(void);
void done_input(void);
int wind_up(void);
int initialize(void);
int handler(void);
void restore_cursor(void);
int iprompt(int, int, int, int, int, int, int);
void write_sim_state(void);
void user_purchase(void);
void user_setup(void);
void user_load(void);
void user_move(void);
int query(void);
int get_modes(void);


static int high, low;

/* Get the user input */
int getkey()
{
     int ch;

     /* normal key codes */
     if ((ch = bdos(KEYIN, 0, 0) & LOBYTE) != '\0')
            return(ch);

     /* convert scan codes to unique internal codes */
     return((bdos(KEYIN, 0, 0) & LOBYTE) | XF);
}
```

```
int initialize()
{
    union REGS inregs, outregs;
    int Vmode, Vpage;

    /* obtain current video mode */
    inregs.h.ah = GET_STATE;
    int86(VIDEO_IO, &inregs, &outregs);
    Vmode = outregs.h.al;
    Vpage = outregs.h.bh;

    /* obtain current bios cursor state */
    inregs.h.ah = GET_CUR;
    inregs.h.bh = Vpage;
    int86(VIDEO_IO, &inregs, &outregs);
    high = outregs.h.ch;
    low = outregs.h.cl;

    /* check to make sure EGA/CGA adapter is available */
    if(Vmode == 7){
        /* The adapter is Monochrome */
        fputs("OPT: This program requires an EGA/CGA adapter.\n", stderr);
        exit(0);
    }

    /* pipe Ctlr-C break signals to handler() */
    if(signal(SIGINT, handler) == (int(*)())-1){
        fputs("OPT: couldn't set SIGINT.\n", stderr);
        exit(0);
    }

    /* make sure we are in mode 3, else change */
    if(Vmode != 3){
        _setvideomode(_TEXTC80);
    }

    /* make sure we are in page 0, else change */
    if(Vpage != 0){
        pageno = 0;
        set_page(pageno);
    }

    /* hide the bios cursor, set bits 4,5 of CH */
    inregs.h.ah = CUR_TYPE;
    inregs.h.ch = high | 0x30;
    inregs.h.cl = low;
    int86(VIDEO_IO, &inregs, &outregs);

    vicls(0);
    vicls(1);
    vicls(2);
```

```
    vicls(3);
}

/* Hide the cursor by setting bits 4 and 5 of CH */
void hide_cursor()
{
    union REGS inregs, outregs;

    inregs.h.ah = CUR_TYPE;
    inregs.h.ch = high | 0x30;
    inregs.h.cl = low;
    int86(VIDEO_ID, &inregs, &outregs);
}

/* Position the cursor at the specified page, row, col */
unsigned int position_cursor(Vpage, row, col)
int Vpage, row, col;
{
    union REGS inregs, outregs;

    inregs.h.ah = CUR_POS;
    inregs.h.bh = (unsigned int)Vpage & 0x07;
    inregs.h.dh = (unsigned int)row & 0xFF;
    inregs.h.dl = (unsigned int)col & 0xFF;
    int86(VIDEO_ID, &inregs, &outregs);

    return(outregs.x.cflag);
}

/* Set visual page */
void set_page(vpage)
int vpage;
{
   union REGS inregs, outregs;

   inregs.h.ah = SET_PAGE;
   inregs.h.al = vpage;
   int86(VIDEO_ID, &inregs, &outregs);
}

/* Map the part type to corresponding character */
int map_part_type(ptype, ptype_in_char)
int ptype;
char *ptype_in_char;
{
    *ptype_in_char = ptype+64;
}

/* Error sensitive malloc */
char *my_malloc(cnt)
int cnt;
```

```
{
    char *ptr;

    if ((ptr = (char*)malloc(cnt)) == NULL){
        fprintf(stderr, "Abort, out of memory!\n");
        exit(2);
    }
    return(ptr);
}

/* Sound a beep at the terminal */
void beep()
{
    putc('\x07', stderr);
}

int wind_up()
{
    union REGS inregs, outregs;

    dispose_calendar();
    vicls(0);
    vicls(1);
    vicls(2);
    vicls(3);
    if(pageno != 0) set_page(0);

    _setvideomode(_DEFAULTMODE);

    /* restore cursor */
    inregs.h.ah = 1;
    inregs.h.ch = high;
    inregs.h.cl = low;
    int86(VIDEO_IO, &inregs, &outregs);
}

/* Restore the BIOS cursor */
void restore_cursor()
{
    union REGS inregs, outregs;

    inregs.h.ah = 1;
    inregs.h.ch = high;
    inregs.h.cl = low;
    int86(VIDEO_IO, &inregs, &outregs);
}

/* Procedure to be invoked if user presses Ctrl-C */
int handler()
{
    char ch;
```

```
    /* Disallow Ctrl-C during handler: */
    viblank(pageno, 23, 0, 23, 79);
    position_cursor(pageno, 23, 0);
    signal(SIGINT, SIG_IGN);
    beep();
    if(!freezed) stop_sim();
    viprintf(pageno, 23, 25, MSG_COLOR, "%s", "Terminate Simulation? (y/n)");
    ch = getch();
    if((ch == 'y') || (ch == 'Y')){
        wind_up();
        exit(0);
    }

    /*
    now call "signal" again so the next interrupt
    signal sends control to handler(), not to
    the operating system.
    */

    signal(SIGINT, handler);
    viblank(pageno, 23, 0, 23, 79);
    if(!freezed) start_sim();
    position_cursor(pageno, 23, 0);
}


/* Get the new break_point from the user */
int get_break_point(old)
int old;
{
    char ch;
    int new;

    viblank(0, 23, 0, 23, 79);
    viprintf(0, 23, 0, MSG_COLOR, "%s", "Break_point:");
    viprintf(0, 23, 13, B_RED+F_WHITE+INT, "%6d", old);
    viprintf(0, 23, 25, MSG_COLOR, "New Break_point: (1 - %d)", MIN_PER_DAY);
    fflush(stdin);
    new = iprompt(0, 23, 53, 4, old, 1, MIN_PER_DAY);
    if(new == old){
        viblank(pageno, 23, 0, 23, 79);
        viprintf(pageno, 23, 30, MSG_COLOR, "%s", "Break_point not changed");
        position_cursor(0, 23, 0);
        hide_cursor();
        return(old);
    }else{
        viblank(0, 23, 0, 23, 79);
        position_cursor(0, 23, 0);
        hide_cursor();
        return(new);
```

```
    ]
}

/* Prompt for an integer value of length 'length' */
int iprompt(Vpage, row, col, length, old, min, max)
int Vpage, row, col, length, old, min, max;
{
    char buf[4];
    int i, j;
    int new;
    int more;

    if(length < 1) return(old);
    restore_cursor();
get_input:
    position_cursor(Vpage, row, col);
    for(i=0; i < length+1; i++){
        fprintf(stderr, "_");
    }
    for(i=0; i < length+1; i++){
        fprintf(stderr, "\b");
    }
    for(i=0, more=TRUE; i < length+1 && more; ){
        buf[i] = getch();
        buf[i+1] = '\0';
        switch(buf[i]){
            case '\r':
            case '\n':
                if(i == 0){
                    return(old);
                }else{
                for(j=0; buf[j] == ' ' || buf[j] == '\t'; j++);
                for(new=0; buf[j] >= '0' && buf[j] <= '9'; j++){
                    new = 10 * new + buf[j] - '0';
                    }
                    if((new <= max) && (new >= min)){
                        return(new);
                    }else{
                        beep();
                        return(old);
                    ]
                }
                break;
            case '\b':
                buf[i] = '\0';
                if(i > 0){
                    buf[--i] = '\0';
                    fputc('\b', stderr);
                    fputc('_', stderr);
                    fputc('_', stderr);
                    fprintf(stderr, "\b\b");
```

```
                        }
                        break;
                default:
                        buf[i+1] = '\0';
                        putc(buf[i++], stderr);
                        if(i > length+1){
                                more = FALSE;
                        }
                        break;
                }
        }
        beep();
        buf[0] = '\0';
        goto get_input;
}

/* Get the user input for writing simulation state to a file */
void write_sim_state()
{
        FILE *fp;
        DEPT *temp_d;
        MACHINE *temp_m;
        PART_INFO *temp_pi;
        RAW *temp_raw;
        PRODUCT *temp_prod;
        char filename[15];
        int i, j, k, more;
        int ext = 0;
        char ch, s = ' ';

        restore_cursor();
        viblank(pageno,23,0,23,79);
        viprintf(pageno,23,0,MSG_COLOR,"%s","Write simulation state to filename[.out]:");
        viprintf(pageno,23,42,B_RED+F_WHITE+INT, "%s", "        ");
        position_cursor(pageno,23,42);
        for(i=0, more=TRUE; i < 8 && more; ){
                filename[i] = getch();
                switch(filename[i]){
                        case '\r':
                        case '\n':
                                if(i == 0){
                                        done_input();
                                        return;
                                }else{
                                        putc(filename[i], stderr);
                                        filename[i] = '\0';
                                        goto proceed;
                                }
                                break;
                        case '\b':
                                filename[i] = '\0';
```

```
                        if(i > 0){
                                filename[--i] = '\0';
                                fputc('\b', stderr);
                        }
                        break;
                default:
                        filename[i+1] = '\0';
                        putc(filename[i], stderr);
                        i = i + 1;
                        if(i > 8){
                                more = FALSE;
                        }
                        break;
            }
        }
proceed:
        for(i=0; filename[i]!='\0'; i++){
            if(filename[i] == '.') ext = 1;
        }
        if(!ext){
            strcat(filename,".out");
        }
        if((fp = fopen(filename, "wt")) == NULL){
            viprintf(pageno,23,55,B_RED+F_WHITE+INT, "%s", "bad file name");
            hide_cursor();
            return;
        }

        fprintf(fp, "%s %10d \n","cash:",cash);
        fprintf(fp, "%s %7d \n","expense:",expense);
        fprintf(fp, "%s %7d \n","n_depts:",n_depts);
        fprintf(fp, "%s %6d \n","n_ptypes:",n_ptypes);
        fprintf(fp, "%s %7d \n","max_ops:",max_ops);
        fprintf(fp, "%s %10d \n","pace:",pace);
        fprintf(fp, "\n");

        fprintf(fp, "%s \n","routing:");
        for(i=0; i<n_ptypes; i++){
            for(j=0; j<max_ops+1; j++){
                fprintf(fp, "%3d",route[i][j]);
            }
            fprintf(fp, "\n");
        }
        fprintf(fp, "\n");

        fprintf(fp, "%s  %5d \n","routing_row:",routing_row);
        fprintf(fp, "%s  %5d \n","routing_col:",routing_col);
        fprintf(fp, "%s  %5d \n","raw_mat_row:",raw_mat_row);
        fprintf(fp, "%s  %5d \n","raw_mat_col:",raw_mat_col);
        fprintf(fp, "%s  %8d \n","prod_row:",prod_row);
        fprintf(fp, "%s  %8d \n","prod_col:",prod_col);
```

```
fprintf(fp, "%s  %7d \n","n_raw_mat:",n_raw_mat);
fprintf(fp, "%s  %7d \n","n_products",n_products);
fprintf(fp, "%s  %4d \n","priority_row:",priority_row);
fprintf(fp, "%s  %4d \n","priority_col:",priority_col);

fprintf(fp, "\n");
for(temp_d = dept_h; temp_d != NULL; temp_d = temp_d->next){
        fprintf(fp, "%s  %9d \n","dept_no:",temp_d->dept_no);
        fprintf(fp, "%s  %7d \n","nmachines:",temp_d->nmachines);
        fprintf(fp, "%s  %4d \n","nmach_in_use:",temp_d->nmach_in_use);
        fprintf(fp, "%s  %13d \n","row:",temp_d->row);
        fprintf(fp, "%s  %13d \n","col:",temp_d->col);
        fprintf(fp, "\n");

        fprintf(fp, "%s \n","machines in this department:");
        fprintf(fp, "%s \n","row:col:mach_no:repeat:setup_for:mstate:usr_stopped:\
                  st_chg_time:idle:setup:busy");
        for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
                fprintf(fp, "%3d%3d%4d%4d%7d%8d%10d %12lu%7lu%4lu%5lu\n",\
                        temp_m->row,temp_m->col,temp_m->mach_no,temp_m->repeat,\
                        temp_m->setup_for,temp_m->mstate,temp_m->user_stopped,\
                        temp_m->state_change_time,temp_m->time_idle,\
                        temp_m->time_setup,temp_m->time_busy);
        }
        fprintf(fp, "\n");
}

fprintf(fp, "\n");
fprintf(fp, "%s \n","part_information:");
fprintf(fp, "%s \n","dept_no:row:col:repeat:ptype:ptype_in_char:pre:post:");
for(temp_d = dept_h; temp_d != NULL; temp_d = temp_d->next){
    for(temp_pi=temp_d->part_info; temp_pi!=NULL; temp_pi=temp_pi->next){
    fprintf(fp, "%7d%3d%3d%3d%6d %c%c%c%c%c%c %4d%3d\n",\
        temp_pi->dept_no,temp_pi->row,temp_pi->col,temp_pi->repeat,temp_pi->ptype,\
        s,s,s,s,s,s,s,temp_pi->ptype_in_char,temp_pi->pre,temp_pi->post);
        }
        fprintf(fp, "\n");
}
fprintf(fp, "\n");


fprintf(fp, "%s \n","raw_materials:");
fprintf(fp, "%s \n","raw_mat_no:cost:row:col");
for(temp_raw = raw_mat; temp_raw != NULL; temp_raw = temp_raw->next){
        fprintf(fp, "%5d %7d %5d %3d \n", temp_raw->raw_mat_no,\
                temp_raw->cost,temp_raw->row,temp_raw->col);
}
fprintf(fp, "\n");

fprintf(fp, "%s \n","products:");
fprintf(fp, "%s \n","prod_no:price:qnty_produced:limit:row:col");
```

```
for(temp_prod = product; temp_prod != NULL; temp_prod = temp_prod->next){
        fprintf(fp, "%5d %5d %9d %9d %4d %4d \n",temp_prod->prod_no,\
                temp_prod->price,temp_prod->qnty_produced,\
                temp_prod->limit,temp_prod->row,temp_prod->col);
}

fprintf(fp, "\n");
fprintf(fp, "\n %s \n","setup_times:");
for(i=0; i<2; i++){
        fprintf(fp, "%s %2d \n","repeat=",i);
        for(j=0; j < n_depts; j++){
                for(k=0; k < n_ptypes; k++){
                        fprintf(fp, "%5d", stime[i][j][k]);
                }
                fprintf(fp, "\n");
        }
        fprintf(fp, "\n");
}

fprintf(fp, "\n");
fprintf(fp, "\n %s \n","machining_times:");
for(i=0; i<2; i++){
        fprintf(fp, "%s %2d \n","repeat=",i);
        for(j=0; j < n_depts; j++){
                for(k=0; k < n_ptypes; k++){
                        fprintf(fp, "%5d", mtime[i][j][k]);
                }
                fprintf(fp, "\n");
        }
        fprintf(fp, "\n");
}

fprintf(fp,"%s\n%s%s\n","part_type:raw_material-relationship:","ptype","rawmat");
for(i=0; i < n_ptypes; i++){
        fprintf(fp, "%3d %8d \n",i+1,raw_for_ptype[i]);
}

fprintf(fp, "\n");
fprintf(fp, "%s %91u \n","time_now:      ",time_now);
fprintf(fp, "%s %9d \n","week_now:      ",week_now);
fprintf(fp, "%s %10d \n","day_now:      ",day_now);
fprintf(fp, "%s %11d \n","hr_now:      ",hr_now);
fprintf(fp, "%s %10d \n","min_now:      ",min_now);

fprintf(fp, "%s %10d \n","deleted:      ",deleted);
fprintf(fp, "%s %9d \n","user_specified:",user_specified);
fprintf(fp, "%s %7d \n","processing:      ",processing);
fprintf(fp, "%s %10d \n","auto_purchase:",auto_purchase);
fprintf(fp, "%s %7d \n","auto_setup:      ",auto_setup);
fprintf(fp, "%s %8d \n","auto_move:      ",auto_move);
fprintf(fp, "%s %9d \n","zoom_row:      ",zoom_row);
```

```
    fprintf(fp, "%s %9d \n","zoom_col:      ",zoom_col);
    fprintf(fp, "%s %8d \n","zoom_dept:     ",zoom_dept);
    fprintf(fp, "%s %10d \n","min_inv:      ",min_inv);
    fprintf(fp, "%s %6d \n","purchase_lot_size:",purchase_lot_size);
    fprintf(fp, "%s %9d \n","transfer_batch:",transfer_batch);
    fprintf(fp, "%s %6d \n","mat_handling_cost:",mat_handling_cost);
    fprintf(fp, "%s %7d \n","other_op_expense:",other_op_expense);

    fprintf(fp, "\n");
    fprintf(fp, "%s \n","statistics_structure:");
        fprintf(fp, "%s  %11d \n","row:        ",stats->fin_row);
        fprintf(fp, "%s  %11d \n","col:        ",stats->fin_col);
        fprintf(fp, "%s  %6ld \n","start_cash:      ",stats->start_cash);
        fprintf(fp, "%s  %8ld \n","cash_now:      ",stats->cash_now);
        fprintf(fp, "%s  %9ld \n","sales_revenue:",stats->sales_revenue);
        fprintf(fp, "%s  %11d \n","raw_mat_exp:",stats->raw_mat_exp);
        fprintf(fp, "%s  %10d \n","mat_hand_exp:",stats->mat_hand_exp);
        fprintf(fp, "%s  %10d \n","other_op_exp:",stats->other_op_exp);
        fprintf(fp, "%s  %9d \n","net_cash_flow:",stats->net_cash_flow);
        fprintf(fp, "%s %71u \n","event_time:      ",event_time);
        fprintf(fp, "%s %5d \n","week:          ",week);
        fprintf(fp, "%s %6d \n","day:           ",day);
        fprintf(fp, "%s %7d \n","hr:            ",hr);
        fprintf(fp, "%s %6d \n","min:           ",min);
        fprintf(fp, "%s %7d \n","event_var1:    ",event_var1);
        fprintf(fp, "%s %7d \n","event_var2:    ",event_var2);
        fprintf(fp, "%s %7d \n","event_var3:    ",event_var3);
        fprintf(fp, "%s %7d \n","event_var4:    ",event_var4);
        fprintf(fp, "%s %s \n","event_type:",event_type);
        fprintf(fp, "%s %c \n","modifier:  ",modifier);
    fclose(fp);
    if(write_cal_list(filename) == 1){
        viprintf(pageno,23,55,B_RED+F_WHITE+INT, "%s", "finished writing");
        hide_cursor();
        return;
    }else{
        hide_cursor();
        return;
    }
}

/* Get the user input for scheduling purchase event */
void user_purchase()
{
    char ch, max_ptype, ptype;
    char pchar;
    int ok, more;
    int i,j,w,d,h,m;
    unsigned long int timex;
    int lot_size, part_no;
```

```
        map_part_type(n_ptypes, &pchar);
        max_ptype = toupper(pchar);
        restore_cursor();
        viblank(pageno, 23, 0, 23, 79);
        viprintf(pageno, 23, 0, MSG_COLOR, "%s", "Purchase raw material for: ");
        position_cursor(pageno, 23, 27);
        ok = FALSE;
        while(!ok){
                ch = getch();
                if(ch < ' '){
                        done_input();
                        return;
                }
                if(ch < 'A' || ch > 'z'){
                        ok = FALSE;
                }else{
                        ok = TRUE;
                }
                if(ok){
                        ptype = toupper(ch);
                        if(ptype <= max_ptype){
                                fputc(ptype, stderr);
                        }else{
                                ok = FALSE;
                        }
                }
        }
get_lot_size:
        lot_size = PURCHASE_LOT_SIZE;
        viprintf(pageno, 23, 30, MSG_COLOR, "%s", "lot size: ");
        viprintf(pageno, 23, 40, B_RED+F_WHITE+INT, "(%3d)", lot_size);
        viprintf(pageno, 23, 55, B_BLACK+F_GREEN+INT, "%s", "Hit Return for default");
        lot_size = iprompt(pageno, 23, 47, 3, lot_size, 1, 999);

        restore_cursor();
        viblank(pageno,23,0,23,79);
        viprintf(pageno,23,0,B_RED+F_WHITE+INT, "%s", "Give schedule time:\
                (Hit return for automatic)");
        viprintf(pageno,23,47,B_BLACK+F_GREEN+INT, "%s", "week:");
        position_cursor(pageno,23,53);
        ok = FALSE;
        while(!ok){
                ch = getch();
                if(ch == '\r'){
                        done_input();
                        goto front;
                }
                if(ch < '0' || ch > '4'){
                        ok = FALSE;
                }else{
                        ok = TRUE;
```

```
                fputc(ch,stderr);
                w = ch - '0';
        }
    }
    viprintf(pageno,23,55,B_BLACK+F_GREEN+INT, "%s", "day:");
    position_cursor(pageno,23,60);
    ok = FALSE;
    while(!ok){
        ch = getch();
        if(ch == '\r'){
            done_input();
            goto front;
        }
        if(ch < '0' || ch > '6'){
            ok = FALSE;
        }else{
            ok = TRUE;
            fputc(ch,stderr);
            d = ch - '0';
        }
    }
    h=0;
    m=0;
    viprintf(pageno,23,62,B_BLACK+F_GREEN+INT, "%s", "hour:");
    h = iprompt(pageno,23,67,2,h,0,23);
    viprintf(pageno,23,70,B_BLACK+F_GREEN+INT, "%s", "min:");
    m = iprompt(pageno,23,74,2,m,0,59);

    string_to_time(&timex,w,d,h,m);
    if(timex <= time_now){
        done_input();
        return;
    }
    goto forward;
front:
    timex = time_now + 1L;
forward:
    if(lot_size > 0){
        part_no = map_char(ptype);
        user_specified = TRUE;
        schedule(PURCHASE, timex,raw_for_ptype[part_no-1],\
            part_no, lot_size, 0);
        user_specified = FALSE;
        done_input();
        return;
    }else{
        goto get_lot_size;
    }
}

/* Get the user input for scheduling setup event */
```

```
void user_setup()
{
     DEPT *temp_d;
     MACHINE *temp_m;
     PART_INFO *temp_pi;
     int ok, same;
     int dept_no, mach_no, repeat, count;
     char part_type;
     char ch;

     restore_cursor();
get_mach:
     viblank(pageno, 23, 0, 23, 79);
     viprintf(pageno, 23, 0, MSG_COLOR, "%s", "Change setup for - dept:");
     viprintf(pageno, 23, 30, MSG_COLOR, "%s", "mach:");
     position_cursor(pageno, 23, 25);
     ok = FALSE;
     while(!ok){
            ch = getch();
            if(ch < ' '){
                   done_input();
                   return;
            }
            if(ch < '1' || ch > '9'){
                   ok = FALSE;
            }else{
                   ok = TRUE;
            }
            if(ok){
                   dept_no = ch - '0';
                   if(dept_no > n_depts){
                          ok = FALSE;
                   }else{
                          ok = TRUE;
                          fputc(ch, stderr);
                   }
            }
     }
     for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no;\
              temp_d = temp_d->next);
     if(temp_d == NULL) print_errord();
     position_cursor(pageno, 23, 36);
     ok = FALSE;
     while(!ok){
            ch = getch();
            if(ch < ' '){
                   done_input();
                   return;
            }
            if(ch < '1' || ch > '9'){
                   ok = FALSE;
```

```
            }else{
                    ok = TRUE;
            }
            if(ok){
                    mach_no = ch - '0';
                    if(mach_no > temp_d->nmachines){
                            ok = FALSE;
                    }else{
                            ok = TRUE;
                            fputc(ch, stderr);
                    }
            }
      }
      for(temp_m = temp_d->mach; temp_m != NULL && temp_m->mach_no != mach_no;\
                  temp_m = temp_m->next);
      if(temp_m == NULL) print_errorm();
      if(temp_m->mstate == 0){
            processing = FALSE;
      }
      if(temp_m->mstate == 1){
            viprintf(pageno, 23, 40, B_BLACK+F_BROWN+INT, "%s",\
                  "Waste previous setup? (y/n) ");
            position_cursor(pageno, 23, 68);
            ch = getch();
            if((ch == 'Y') || (ch == 'y')){
                    viblank(pageno, 23, 40, 23, 79);
                    goto get_ptype;
            }else{
                    goto get_mach;
            }
      }
      if(temp_m->mstate == 2){
            processing = TRUE;
      }
get_ptype:
      viprintf(pageno, 23, 40, MSG_COLOR, "%s", "for part_type:");
      position_cursor(pageno, 23, 55);
      ok = FALSE;
      while(!ok){
            ch = getch();
            if(ch < ' '){
                    done_input();
                    return;
            }
            if(ch < 'A' || ch > 'z'){
                    ok = FALSE;
            }else{
                    ok = TRUE;
            }
            if(ok){
                    part_type = toupper(ch);
```

```
                same = TRUE;
                for(temp_pi = temp_d->part_info; temp_pi != NULL && same; ){
                        if(temp_pi->ptype_in_char == part_type){
                                same = FALSE;
                        }else{
                                temp_pi = temp_pi->next;
                        }
                }
                if(temp_pi == NULL){
                        ok = FALSE;
                }else{
                        ok = TRUE;
                        fputc(part_type, stderr);
                }
        }
}
viprintf(pageno, 23, 63, MSG_COLOR, "%s", "repeat:");
position_cursor(pageno, 23, 71);
ok = FALSE;
while(!ok){
        ch = getch();
        if(ch < ' '){
                done_input();
                return;
        }
        if(ch < '0' || ch > '9'){
                ok = FALSE;
        }else{
                ok = TRUE;
        }
        if(ok){
                repeat = ch - '0';
                for(count=0;\
                        temp_pi != NULL && count < repeat;\
                                temp_pi = temp_pi->next, count++);
                if(temp_pi == NULL){
                viprintf(pageno, 22, 40, MSG_COLOR, "%s", "Illegal part_type:");
                viprintf(pageno, 22, 59, B_BLACK+F_GREEN+INT, "%ld", repeat);
                viprintf(pageno, 22, 60, B_BLACK+F_BROWN+INT, "%c", part_type);
                        beep();
                        viblank(pageno, 23, 55, 23, 79);
                        ok = FALSE;
                        goto get_ptype;
                }
                if((strcmp(temp_pi->ptype_in_char, part_type) == 0) && \
                        (temp_pi->repeat == repeat)){
                        change(SETUP,mach_no,dept_no,repeat,map_char(part_type));
                        fputc(ch, stderr);
                }else{
                viprintf(pageno, 22, 40, MSG_COLOR, "%s", "Illegal part_type:");
                viprintf(pageno, 22, 59, B_BLACK+F_GREEN+INT, "%ld", repeat);
```

```
                    viprintf(pageno, 22, 60, B_BLACK+F_BROWN+INT, "%c", part_type);
                            beep();
                            viblank(pageno, 23, 55, 23, 79);
                            ok = FALSE;
                            goto get_ptype;
                    }
            }
      }
      done_input();
      if(temp_m->setup_for == UN_UTILIZED){
            if(temp_d->nmach_in_use < temp_d->nmachines){
                  temp_d->nmach_in_use++;
            }
      }
}

/* Get the user input for scheduling load event */
void user_load()
{
      DEPT *temp_d;
      MACHINE *temp_m;
      PART_INFO *temp_pi;
      char ch;
      int ok, found;
      int dept_no, mach_no;
      int part_type, repeat;

      restore_cursor();
      viblank(pageno, 23, 0, 23, 79);
      viprintf(pageno, 23, 0, MSG_COLOR, "%s", "Load - Dept:      Mach:");
get_load_mach:
      ok = FALSE;
      position_cursor(pageno, 23, 13);
      while(!ok){
            ch = getch();
            if(ch < ' '){
                  done_input();
                  return;
            }
            if(ch < '1' || ch > '9'){
                  ok = FALSE;
            }else{
                  ok = TRUE;
            }
            if(ok){
                  dept_no = ch - '0';
                  if(dept_no > n_depts){
                        ok = FALSE;
                  }else{
                        ok = TRUE;
                        fputc(ch, stderr);
```

```
                }
        }
}
for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no; \
        temp_d = temp_d->next);
if(temp_d == NULL) print_errord();
position_cursor(pageno, 23, 24);
ok = FALSE;
while(!ok){
        ch = getch();
        if(ch < ' '){
                done_input();
                return;
        }
        if(ch < '1' || ch > '9'){
                ok = FALSE;
        }else{
                ok = TRUE;
        }
        if(ok){
                mach_no = ch - '0';
                if(mach_no > temp_d->nmachines){
                        ok = FALSE;
                }else{
                        ok = TRUE;
                        fputc(ch, stderr);
                }
        }
}
for(temp_m = temp_d->mach; temp_m != NULL && temp_m->mach_no != mach_no; \
        temp_m = temp_m->next);
if(temp_m == NULL) print_errorm();
viblank(pageno, 22, 0, 22, 79);
if(temp_m->setup_for == UN_UTILIZED){
        done_input();
        viprintf(pageno, 22, 0, MSG_COLOR, "%s", "Setup machine before loading");
        return;
}
if(temp_m->mstate != 0){
        viprintf(pageno, 22, 0, MSG_COLOR, "%s", "Machine is not free");
        ok = FALSE;
        goto get_load_mach;
}
part_type = temp_m->setup_for;
repeat = temp_m->repeat;
for(temp_pi = temp_d->part_info;\
        temp_pi != NULL && !((temp_pi->ptype == part_type) &&\
                (temp_pi->repeat == repeat));\
                        temp_pi = temp_pi->next);
if(temp_pi == NULL) print_errorp();
if(temp_pi->pre > 0){
```

```
            change(LOAD, mach_no, dept_no, repeat, part_type);
        }
        done_input();
        if(temp_pi->pre <= 0){
            viprintf(pageno, 23, 25, B_BLACK+F_BROWN+INT, "No parts in the pre area");
        }
        return;
}


/* Get the user input for scheduling move event */
void user_move()
{

        DEPT *temp_d;
        PART_INFO *temp_pi, *temp_pi_dummy;
        PRODUCT *temp_prod;
        int part_no, dept_no, repeat, size, count;
        int j, ok, dont, found, last;
        int m, d, h, m;
        unsigned long int timex;
        char ch, max_ptype, parttype;
        char pchar;

        map_part_type(n_ptypes, &pchar);
        max_ptype = toupper(pchar);
        dont = TRUE;
        last = FALSE;
        restore_cursor();
        viblank(pageno,23,0,23,79);
        viprintf(pageno,23,0,B_BLACK+F_GREEN+INT,"%s","Move- Part_type:   from dept:");
        position_cursor(pageno, 23, 17);
        ok = FALSE;
        while(!ok){
            ch = getch();
            if(ch < ' '){
                done_input();
                return;
            }
            if(ch < 'A' || ch > 'z'){
                ok = FALSE;
            }else{
                ok = TRUE;
            }
            if(ok){
                parttype = toupper(ch);
                if(parttype <= max_ptype){
                    fputc(parttype, stderr);
                }else{
                    ok = FALSE;
                }
```

```
        }
}

position_cursor(pageno, 23, 30);
ok = FALSE;
while(!ok){
        ch = getch();
        if(ch < ' '){
                done_input();
                return;
        }
        if(ch < '1' || ch > '9'){
                ok = FALSE;
        }else{
                ok = TRUE;
        }
        if(ok){
                dept_no = ch - '0';
                if(dept_no > n_depts){
                        ok = FALSE;
                }else{
                        part_no = map_char(parttype);
                        found = FALSE;
                        for(j=0;j<=(max_ops+1)) && (dont==TRUE);j++){
                                if(route[part_no-1][j]==dept_no){
                                        if(route[part_no-1][j+1]==0){
                                                found = TRUE;
                                                last = TRUE;
                                                ok = TRUE;
                                                found = TRUE;
                                                fputc(ch,stderr);
                                                dont = FALSE;
                                        }else{
                                                ok = TRUE;
                                                found = TRUE;
                                                fputc(ch,stderr);
                                                dont = FALSE;
                                        }
                                }
                        }
                }
        }
        if(found == FALSE){
                ok = FALSE;
        }
}

for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no; \
                temp_d = temp_d->next);
if(temp_d == NULL) print_errord;
for(temp_pi = temp_d->part_info;\
```

```
        temp_pi != NULL && (strcmp(temp_pi->ptype_in_char,parttype) != 0);\
                temp_pi = temp_pi->next);
for(temp_prod = product; temp_prod->prod_no != part_no;\
                temp_prod = temp_prod->next);

if(last == TRUE){
        if(temp_prod->limit-temp_prod->qnty_produced == 0){
                viprintf(pageno,23,35,B_RED+F_WHITE+INT,"%s","Exceeds sales limit");
                hide_cursor();
                return;
        }
}

viprintf(pageno, 23, 32, MSG_COLOR, "%s", "repeat:");
position_cursor(pageno, 23, 40);
ok = FALSE;
while(!ok){
        ch = getch();
        if(ch < ' '){
                done_input();
                return;
        }
        if(ch < '0' || ch > '9'){
                ok = FALSE;
        }else{
                ok = TRUE;
        }
        if(ok){
                repeat = ch - '0';
                if(repeat == 0){
                        viblank(pageno,22,0,22,79);
                }
                temp_pi_dummy = temp_pi;
                for(count=0;\
                        temp_pi != NULL && count < repeat;\
                                temp_pi = temp_pi->next, count++);
                if(temp_pi == NULL){
                viprintf(pageno, 22, 40, MSG_COLOR, "%s", "No repeat for this part_type");
                        beep();
                        ok = FALSE;
                }
                if((strcmp(temp_pi->ptype_in_char, parttype) == 0) && \
                            (temp_pi->repeat == repeat)){
                        fputc(ch, stderr);
                        viblank(pageno,22,0,22,79);
                }else{
                viprintf(pageno, 22, 40, MSG_COLOR, "%s", "No repeat for this part_type");
                        beep();
                        ok = FALSE;
                        temp_pi = temp_pi_dummy;
                }
```

```
            }
        }

        viprintf(pageno,23,42,MSG_COLOR,"%s","Quantity:");
    getsize:
        position_cursor(pageno,23,52);
        size = 0;
        size = iprompt(pageno,23,52,2,size,0,99);
        viblank(pageno,22,0,22,79);
        if(size == 0){
                done_input();
                return;
        }
        if(temp_pi->post == 0){
                done_input();
                return;
        }
        if(temp_pi->post >= size){
                if(last == TRUE){
                        if( size >= temp_prod->limit - temp_prod->qnty_produced){
                                size = temp_prod->limit - temp_prod->qnty_produced;
                        }
                }
                goto gettime;
        }else{
        viprintf(pageno, 22, 40, B_RED+F_WHITE+INT, "%s", "Not enough parts in post area");
                viblank(pageno,23,52,23,79);
                goto getsize;
        }
    gettime:
        restore_cursor();
        viblank(pageno,23,0,23,79);
        viprintf(pageno,23,0,B_RED+F_WHITE+INT, "%s", \
                 "Give schedule time: (Hit return for automatic)");
        viprintf(pageno,23,47,B_BLACK+F_GREEN+INT, "%s", "week:");
        position_cursor(pageno,23,53);
        ok = FALSE;
        while(!ok){
                ch = getch();
                if(ch == '\r'){
                        done_input();
                        goto front;
                }
                if(ch < '0' || ch > '4'){
                        ok = FALSE;
                }else{
                        ok = TRUE;
                        fputc(ch,stderr);
                        w = ch - '0';
                }
        }
```

```
    viprintf(pageno,23,55,B_BLACK+F_GREEN+INT, "%s", "day:");
    position_cursor(pageno,23,60);
    ok = FALSE;
    while(!ok){
            ch = getch();
            if(ch == '\r'){
                    done_input();
                    goto front;
            }
            if(ch < '0' || ch > '6'){
                    ok = FALSE;
            }else{
                    ok = TRUE;
                    fputc(ch,stderr);
                    d = ch - '0';
            }
    }
    h=0;
    m=0;
    viprintf(pageno,23,62,B_BLACK+F_GREEN+INT, "%s", "hour:");
    h = iprompt(pageno,23,67,2,h,0,23);
    viprintf(pageno,23,70,B_BLACK+F_GREEN+INT, "%s", "min:");
    m = iprompt(pageno,23,74,2,m,0,59);

    string_to_time(&timex,w,d,h,m);
    if(timex <= time_now){
            done_input();
            return;
    }
    goto forward;
front:
    timex = time_now + 1L;
forward:
    user_specified = TRUE;
    schedule(MOVE,timex,repeat,part_no,dept_no,size);
    temp_pi->post = temp_pi->post - size;
    user_specified = FALSE;
    done_input();
    return;
}

void done_input(void)
{
    position_cursor(pageno, 23, 0);
    hide_cursor();
    viblank(pageno, 22, 0, 22, 79);
    viblank(pageno, 23, 0, 23, 79);
}

/* Map the part type to corresponding integer */
int map_char(ch)
```

```
char ch;
{
    return(ch-64);
}

int query()
{
    char ch;

    set_page(1);
    if(state == FALSE){
        auto_purchase = TRUE;
    }
    viprintf(1, 23, 13, MSG_COLOR, "%s", \
        "Do you want automatic purchase of raw materials? <y/n>");
    ch = getch();
    if((ch == 'n') || (ch == 'N')){
        auto_purchase = FALSE;
    }
    viblank(1, 23, 0, 23, 79);
    if(state == FALSE){
        auto_setup = TRUE;
    }
    viprintf(1, 23, 25, MSG_COLOR, "%s", \
        "Do you want automatic setups? <y/n>");
    ch = getch();
    if((ch == 'n') || (ch == 'N')){
        auto_setup = FALSE;
    }
    viblank(1, 23, 0, 23, 79);
    if(state == FALSE){
        auto_move = TRUE;
    }
    viprintf(1, 23, 15, MSG_COLOR, "%s", \
        "Do you want automatic material transfers? <y/n>");
    ch = getch();
    if((ch == 'n') || (ch == 'N')){
        auto_move = FALSE;
    }
    viblank(1, 23, 0, 23, 79);
    viprintf(1, 23, 13, MSG_COLOR, "%s", \
        "Do you want to change any of the above variables? <y/n>");
    ch = getch();
    if((ch == 'y') || (ch == 'Y')){
        change_vars();
    }
    viblank(1, 23, 0, 23, 79);
    set_page(0);
    viblank(0, 23, 0, 23, 79);
    viprintf(0, 23, 18, MSG_COLOR, "%s", \
        "You can change the modes by pressing ALT-M");
```

```
}


int get_modes()
{
    char ch;

    viblank(0, 23, 0, 23, 79);
    viprintf(0, 23, 15, MSG_COLOR, "%s", \
            "Press space bar to toggle mode or return to accept");
    vichgatt(0, 16, 3, 6, BLNK);
get_p_mode:
    ch = getch();
    if((ch == ' ') || (ch == K_RETURN)){
            vichgatt(0, 16, 3, 6, BLNK);
            if(ch == ' '){
                    if(auto_purchase){
                            auto_purchase = FALSE;
                    }else{
                            auto_purchase = TRUE;
                    }
                    print_modes();
                    vichgatt(0, 16, 3, 6, BLNK);
                    goto get_p_mode;
            }else{
                    vichgatt(0, 18, 3, 6, BLNK);
                    goto get_s_mode;
            }
    }else{
            goto get_p_mode;
    }
get_s_mode:
    ch = getch();
    if((ch == ' ') || (ch == K_RETURN)){
            vichgatt(0, 18, 3, 6, BLNK);
            if(ch == ' '){
                    if(auto_setup){
                            auto_setup = FALSE;
                    }else{
                            auto_setup = TRUE;
                    }
                    print_modes();
                    vichgatt(0, 18, 3, 6, BLNK);
                    goto get_s_mode;
            }else{
                    vichgatt(0, 20, 3, 6, BLNK);
                    goto get_a_mode;
            }
    }else{
            goto get_s_mode;
    }
```

```
get_m_mode:
    ch = getch();
    if((ch == ' ') || (ch == K_RETURN)){
            vichgatt(0, 20, 3, 6, BLNK);
            if(ch == ' '){
                    if(auto_move){
                            auto_move = FALSE;
                    }else{
                            auto_move = TRUE;
                    }
                    print_modes();
                    vichgatt(0, 20, 3, 6, BLNK);
                    goto get_m_mode;
            }else{
                    viblank(0, 23, 0, 23, 79);
                    return;
            }
    }else{
            goto get_m_mode;
    }
}
```

```
#include <stdio.h>
#include "user\keydefs.h"
#include "user\opt.h"
#include "user\std.h"
#include "user\vi.h"

int zoom(int);
int get_dept_no(int);
int set_zoom_scr(void);
int set_act_window(int);
int update_zoom_act(int);
int stop_zoom_dept(void);
int start_zoom_dept(void);

/* Function to zoom the specified department */
int zoom(dept_no)
int dept_no;
{
    DEPT *temp_d;
    MACHINE *temp_m;
    PART_INFO *temp_pi;
    int row, col;

    row = zoom_row;
    col = zoom_col;
    viblank(1, 2, 23, 14, 55);
    viprintf(1, row-2, col-5, TITLE_COLOR, "%s %1d", "Department", dept_no);
    for(temp_d = dept_hj; temp_d->dept_no != dept_no; temp_d = temp_d->next);
    for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
        viprintf(1, row, col-1, B_RED+F_WHITE+INT, " %1d ", temp_m->mach_no);
        viprintf(1, row+1, col-3, MACHINE_COLOR, "%s", "      ");
        viprintf(1, row+1, col-3, B_BLUE+F_BROWN+INT, "%c", '\xB3');
        viprintf(1, row+1, col+3, B_BLUE+F_BROWN+INT, "%c", '\xB3');
        switch (temp_m->mstate){
            case 0:
                if(temp_m->setup_for != UNUTILIZED){
                    if(temp_m->repeat > 0){
                        viprintf(1, row+1, col-2, B_BLUE+F_GREEN+INT,\
                            "%1d", temp_m->repeat);
                    }
                viprintf(1,row+1,col-1,MACHINE_COLOR,"%c",temp_m->setup_for_in_char);
                viprintf(1, row+1, col+1, MACHINE_COLOR, "%c", IDLE);
                }
                break;
            case 1:
                if(temp_m->repeat > 0){
                viprintf(1,row+1,col-2,B_BLUE+F_GREEN+INT,"%1d",temp_m->repeat);
                }
                viprintf(1,row+1,col-1,MACHINE_COLOR,"%c",temp_m->setup_for_in_char);
                viprintf(1, row+1, col+1, MACHINE_COLOR+BLNK, "%c", SETTING_UP);
                break;
```

```
                case 2:
                        if(temp_m->repeat > 0){
                        viprintf(1,row+1,col-2,B_BLUE+F_GREEN+INT,"%1d",temp_m->repeat);
                        }
                viprintf(1,row+1,col-1,MACHINE_COLOR,"%c",temp_m->setup_for_in_char);
                viprintf(1, row+1, col+1, MACHINE_COLOR+BLNK, "%c", PROCESSING);
                        break;
        }
        row = row+2;
    }
    row = zoom_row+1;
    col = zoom_col-13;
    for(temp_pi = temp_d->part_info; temp_pi != NULL; temp_pi = temp_pi->next){
        if(temp_pi->repeat > 0){
        viprintf(1, row, col-3, B_BLACK+F_GREEN+INT, "%d", temp_pi->repeat);
        viprintf(1, row, col+28, B_BLACK+F_GREEN+INT, "%d", temp_pi->repeat);
        }
        viprintf(1,row,col-2,B_BLACK+F_BROWN+INT,"%c",temp_pi->ptype_in_char);
        viprintf(1,row,col+29,B_BLACK+F_BROWN+INT,"%c",temp_pi->ptype_in_char);
        if(temp_pi->pre > 0){
                viprintf(1, row, col-1, PRE_COLOR, " %3d ", temp_pi->pre);
        }else{
                viprintf(1, row, col-1, PRE_COLOR, "%s", "     ");
        }
        if(temp_pi->post > 0){
                viprintf(1, row, col+23, POST_COLOR, " %3d ", temp_pi->post);
        }else{
                viprintf(1, row, col+23, POST_COLOR, "%s", "     ");
        }
        viprintf(1, row, col+4, STIME_COLOR, " %3d ",\
                stime[temp_pi->repeat][temp_pi->dept_no-1][temp_pi->ptype-1]);
        viprintf(1, row, col+18, MTIME_COLOR, " %3d ",\
                atime[temp_pi->repeat][temp_pi->dept_no-1][temp_pi->ptype-1]);
        row++;
    }
    set_act_window(zoom_dept);
    update_zoom_act(zoom_dept);
    if(freezed){
        stop_zoom_dept();
    }
    if(state == TRUE){
        update_zoom_clock();
    }
}

/* Function to get the department number if F2 is pressed from zoom window */
int get_dept_no(old)
int old;
{
    int dept_no, ok;
    char ch;
```

```
        restore_cursor();
        viblank(1, 23, 0, 23, 79);
        viprintf(1, 23, 30, MSG_COLOR, "%s", "Department Number:");
get_dept:
        position_cursor(1, 23, 49);
        ok = FALSE;
        while(!ok){
                ch = getch();
                if(ch < ' '){
                        done_input();
                        return(old);
                }
                if(ch < '1' || ch > '9'){
                        ok = FALSE;
                }else{
                        ok = TRUE;
                }
                if(ok){
                        dept_no = ch - '0';
                        if(dept_no > n_depts){
                                ok = FALSE;
                        }else{
                                ok = TRUE;
                                fputc(ch, stderr);
                        }
                }
        }
        done_input();
        return(dept_no);
}

/* Function to set the initial zoom screen */
int set_zoom_scr()
{
        DEPT *temp_d;
        MACHINE *temp_m;
        PART_INFO *temp_pi;
        RAW *temp_raw;
        PRODUCT *temp_prod;
        int row, col;
        char pchar;

        viprintf(1, 1, 3, S_TEXT_COLOR, "%s", "WEEK:     DAY:");
        viprintf(1, 3, 5, S_TEXT_COLOR, "%s", "TIME:");
        viprintf(1, 1, 9, CLOCK_COLOR, "%s", "00");
        viprintf(1, 1, 18, CLOCK_COLOR, "%s", "0");
        viprintf(1, 3, 11, CLOCK_COLOR, "%s", "00:00");
        viprintf(1, 6, 3, S_TEXT_COLOR, "%s", "Shop works for:");
        viprintf(1, 8, 3, TITLE_COLOR, "%s", "hrs per day");
        viprintf(1, 8, 17, B_RED+F_WHITE+INT, "%2d", hrs_per_day);
```

```
        viprintf(1,10, 3, TITLE_COLOR, "%s", "days per week");
        viprintf(1,10, 18, B_RED+F_WHITE+INT, "%1d", days_per_week);
        viprintf(1, 3, 62, TITLE_COLOR, "%s", "Min inventory");
        viprintf(1, 3, 76, B_RED+F_WHITE+INT, " %2d", min_inv);
        viprintf(1, 5, 58, TITLE_COLOR, "%s", "Purchase lot size");
        viprintf(1, 5, 76, B_RED+F_WHITE+INT, "%3d", purchase_lot_size);
        viprintf(1, 7, 61, TITLE_COLOR, "%s", "Transfer batch");
        viprintf(1, 7, 76, B_RED+F_WHITE+INT, "%3d", transfer_batch);
        viprintf(1, 9, 58, TITLE_COLOR, "%s", "Mat handling cost");
        viprintf(1, 9, 76, B_RED+F_WHITE+INT, "%3d", mat_handling_cost);
        viprintf(1, 11, 60, TITLE_COLOR, "%s", "Weekly expense");
        viprintf(1, 11, 75, B_RED+F_WHITE+INT, "%4d", other_op_expense);
        viprintf(1, 13, 2, S_TEXT_COLOR, "%s", "Raw material cost");
        row = 15;
        col = 4;
        for(temp_raw = raw_mat; temp_raw != NULL; temp_raw = temp_raw->next, row++){
                map_part_type(temp_raw->raw_mat_no, &pchar);
                viprintf(1, row, col, TITLE_COLOR, "  %c - ", pchar);
                viprintf(1, row, col+6, B_BLACK+F_CYAN+INT, "$%4d", temp_raw->cost);
        }
        viprintf(1, 13, 64, S_TEXT_COLOR, "%s", "Product price");
        row = 15;
        col = 66;
        for(temp_prod = product; temp_prod != NULL; temp_prod = temp_prod->next, row++){
                map_part_type(temp_prod->prod_no, &pchar);
                viprintf(1, row, col, TITLE_COLOR, "%c -", pchar);
                viprintf(1, row, col+4, B_BLACK+F_CYAN+INT, "$%4d", temp_prod->price);
        }
        zoom(zoom_dept);
}

int set_act_window(dept_no)
int dept_no;
{
        DEPT *temp_d;
        MACHINE *temp_m;
        int row, col;

        for(row=15; row < 22; row++){
                for(col=23; col < 57; col++){
                        viputc(1, row, col, STAT_ACT_COLOR, ' ');
                }
        }
        viprintf(1,15,24,B_BLUE+F_BROWN+INT,"Department%1d - Activation Report",dept_no);
        viprintf(1, 16, 26, STAT_ACT_COLOR, "%s", "Mach#   Prod%  Setup%  Idle%");
        for(temp_d=dept_h; temp_d->dept_no != dept_no; temp_d = temp_d->next);
        for(temp_m=temp_d->mach, row=17; temp_m != NULL; temp_m = temp_m->next,row++){
                viprintf(1, row, 28, STAT_ACT_COLOR, "%1d", temp_m->mach_no);
        }
}
```

```
/* Function to update Activation Report in the zoom window */
int update_zoom_act(dept_no)
int dept_no;
{
     unsigned long int total_time;
     unsigned long int prod_time, setup_time, idle_time;
     int prod, setup, idle;
     int row, col;
     DEPT *temp_d;
     MACHINE *temp_m;

     row = 17;
     col = 35;
     for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != dept_no; \
                    temp_d = temp_d->next);
     for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
            total_time = time_now;
            if (total_time != 0){
                   setup_time = temp_m->time_setup;
                   prod_time = temp_m->time_busy;
                   if(temp_m->mstate == 1){
                          setup_time += (time_now - temp_m->state_change_time);
                   }
                   if(temp_m->mstate == 2){
                          prod_time += (time_now - temp_m->state_change_time);
                   }
                   setup = (int)(((double)setup_time/(double)total_time)*100L);
                   prod = (int)(((double)prod_time/(double)total_time)*100L);
                   idle = 100-(prod+setup);
                   viprintf(1, row, col, STAT_ACT_COLOR, "%3d", prod);
                   viprintf(1, row, col+7, STAT_ACT_COLOR, "%3d", setup);
                   viprintf(1, row, col+15, STAT_ACT_COLOR, "%3d", idle);
            }
            row++;
     }
}

int stor_zoom_dept()
{
     DEPT *temp_d;
     MACHINE *temp_m;
     int row, col;

     row = zoom_row+1;
     col = zoom_col+1;
     for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != zoom_dept; \
                    temp_d = temp_d->next);
     for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
            if((temp_m->mstate == 1) || (temp_m->mstate == 2)){
                   vichgatt(1, row, col, 1, BLNK);
            }
```

```
            row = row+2;
        }
}

int start_zoom_dept()
{
        DEPT *temp_d;
        MACHINE *temp_m;
        int row, col;

        row = zoom_row+1;
        col = zoom_col+1;
        for(temp_d = dept_h; temp_d != NULL && temp_d->dept_no != zoom_dept; \
                        temp_d = temp_d->next);
        for(temp_m = temp_d->mach; temp_m != NULL; temp_m = temp_m->next){
                if((temp_m->mstate == 1) || (temp_m->mstate == 2)){
                        vichgatt(1, row, col, 1, BLNK);
                }
                row = row+2;
        }
}
```

INDEX FOR PROJECT FILES

APPENDIX "B"

SAMPLE INPUT FILES

```
cash:                25000
expense:     0
n_depts:     5
n_ptypes:    5
max_ops:     6

routing:
1 3 5 2 0 0 0
3 4 5 2 1 2 0
1 5 2 5 3 4 0
1 3 5 4 0 0 0
2 5 4 3 1 2 0

routing_row: 0
routing_col: 42

dept_no:     1
nmachines:   3
nmach_in_use:        0
row:         1
col:         21

dept_no:     2
nmachines:   3
nmach_in_use:        0
row:         14
col:         21

dept_no:     3
nmachines:   3
nmach_in_use:        0
row:         9
col:         45

dept_no:     4
nmachines:   3
nmach_in_use:        0
row:         1
col:         68

dept_no:     5
nmachines:   3
nmach_in_use:        0
row:         14
col:         68

raw_materials:
row:         8
col:         11

n_raw_mat:   5
```

```
raw_material_cost:
1    20
2    30
3    15
4    25
5    10

products:
row:          8
col:          60

n_products:  5

product_prices:
1    80      5
2    100     10
3    90      20
4    110     25
5    60      10

setup_times:
repeat=0
240 120 320 160 180
 80 120 220   0 130
 60 210 100 225 130
  0 110 120 115 220
 60 220 110 135 130

repeat=1
  0   0   0   0   0
  0 125   0   0 165
  0   0   0   0   0
  0   0   0   0   0
115   0 100   0   0

machining_times:
repeat=0
  5  12  18   8   6
 50  43  16   0  10
 16   5  17  23  28
  0  12  26  36  18
  8  11  21  31  26

repeat=1
  0   0   0   0   0
  0   2   0   0   8
  0   0   0   0   0
  0   0   0   0   0
 16   0  12   0   0
```

```
part_type:raw_material-relationship:
ptype        rawmat
1    1
2    2
3    3
4    4
5    5
```

```
cash:       22108
expense:     2892
n_depts:        5
n_ptypes:       5
max_ops:        6
pace:           5

routing:
  1  3  5  2  0  0  0
  3  4  5  2  1  2  0
  1  5  2  5  3  4  0
  1  3  5  4  0  0  0
  2  5  4  3  1  2  0

routing_row:        0
routing_col:       42
raw_mat_row:        8
raw_mat_col:       11
prod_row:           8
prod_col:          60
n_raw_mat:          5
n_products          5
priority_row:       0
priority_col:       0

dept_no:            1
nmachines:          3
nmach_in_use:       3
row:                1
col:               21

machines in this department:
row:col:mach_no:repeat:setup_for:mstate:usr_stopped:st_chg_time:idle:setup:busy
  1  21     1      0         1       2           0          345     0   240   105
  3  21     2      0         3       2           0          338     1   320    17
  5  21     3      0         4       2           0          344    19   160   165

dept_no:            2
nmachines:          3
nmach_in_use:       1
row:               14
col:               21

machines in this department:
row:col:mach_no:repeat:setup_for:mstate:usr_stopped:st_chg_time:idle:setup:busy
 14  21     1      0         5       2           0          340    21   130   189
 16  21     2      0       224       0           0            0     0     0     0
 18  21     3      0       224       0           0            0     0     0     0

dept_no:            3
nmachines:          3
```

```
nmach_in_use:     3
row:              9
col:             45

machines in this department:
row:col:mach_no:repeat:setup_for:mstate:usr_stopped:st_chg_time:idle:setup:busy
  9  45    1      0       2        2          0          345       0  210   135
 11  45    2      0       4        1          0          176       0    0     0
 13  45    3      0       1        2          0          342       2   60    30

dept_no:          4
nmachines:        3
nmach_in_use:     2
row:              1
col:             68

machines in this department:
row:col:mach_no:repeat:setup_for:mstate:usr_stopped:st_chg_time:idle:setup:busy
  1  68    1      0       2        2          0          342       0  110    12
  3  68    2      0       5        1          0          332       0    0     0
  5  68    3      0     224        0          0            0       0    0     0

dept_no:          5
nmachines:        3
nmach_in_use:     2
row:             14
col:             68

machines in this department:
row:col:mach_no:repeat:setup_for:mstate:usr_stopped:st_chg_time:idle:setup:busy
 14  68    1      0       5        2          0          332       2  130    50
 16  68    2      0       1        1          0          342       0    0     0
 18  68    3      0     224        0          0            0       0    0     0


part_information:
dept_no:row:col:repeat:ptype:ptype_in_char:pre:post:
      1   2  12      0      1             A    4    1
      1   3  12      0      2             B    0    0
      1   4  12      0      3             C    4    1
      1   5  12      0      4             D    7    1
      1   6  12      0      5             E    0    0

      2  15  12      0      1             A    0    0
      2  16  12      0      2             B    0    0
      2  17  12      1      2             B    0    0
      2  18  12      0      3             C    0    0
      2  19  12      0      5             E    4    1
      2  20  12      1      5             E    0    0

      3  10  36      0      1             A   18    0
```

```
      3  11  36   0      2              B    3   1
      3  12  36   0      3              C    0   0
      3  13  36   0      4              D   22   0
      3  14  36   0      5              E    0   0

      4   2  59   0      2              B   25   1
      4   3  59   0      3              C    0   0
      4   4  59   0      4              D    0   0
      4   5  59   0      5              E    2   0

      5  15  59   0      1              A    2   0
      5  16  59   0      2              B    0   0
      5  17  59   0      3              C    0   0
      5  18  59   1      3              C    0   0
      5  19  59   0      4              D    0   0
      5  20  59   0      5              E   18   0


raw_materials:
raw_mat_no:cost:row:col
    1      20    9  11
    2      30   10  11
    3      15   11  11
    4      25    9  21
    5      10   10  21

products:
prod_no:price:qnty_produced:limit:row:col
    1    80          0        5    9   60
    2   100          0       10   10   60
    3    90          0       20   11   60
    4   110          0       25    9   70
    5    60          0       10   10   70


 setup_times:
repeat=  0
  240  120  320  160  180
   80  120  220    0  130
   60  210  100  225  130
    0  110  120  115  220
   60  220  110  135  130

repeat=  1
    0    0    0    0    0
    0  125    0    0  165
    0    0    0    0    0
    0    0    0    0    0
  115    0  100    0    0
```

```
 machining_times:
repeat=  0
    5   12   18    8    6
   50   43   16    0   10
   16    5   17   23   28
    0   12   26   36   18
    8   11   21   31   26

repeat=  1
    0    0    0    0    0
    0    2    0    0    8
    0    0    0    0    0
    0    0    0    0    0
   16    0   12    0    0

part_type:raw_material-relationship:
 ptype rawmat
   1       1
   2       2
   3       3
   4       4
   5       5

time_now:              345
week_now:                0
day_now:                 0
hr_now:                  5
min_now:                45
deleted:                 0
user_specified:          0
processing:              0
auto_purchase:           1
auto_setup:              1
auto_move:               1
zoom_row:                4
zoom_col:               39
zoom_dept:               1
min_inv:                 2
purchase_lot_size:       5
transfer_batch:          2
mat_handling_cost:       1
other_op_expense:      200

statistics_structure:
row:                     5
col:                    27
start_cash:          25000
cash_now:            22108
sales_revenue:           0
raw_mat_exp:          2475
```

```
mat_hand_exp:          46
other_op_exp:           0
net_cash_flow:          0
event_time:           346
week:                   0
day:                    0
hr:                     5
min:                   46
event_var1:             0
event_var2:             0
event_var3:             0
event_var4:             0
event_type: clock_tick
modifier:   $
```

CALENDAR_LIST:

modifier:	time:	week:	day:	hrs:	mins:	event_function_ptr:	a	b	c	d
$	346	0	0	5	46	clock_tick	0	0	0	0
$	347	0	0	5	47	clock_tick	0	0	0	0
$	348	0	0	5	48	clock_tick	0	0	0	0
$	349	0	0	5	49	clock_tick	0	0	0	0
$	350	0	0	5	50	unload	1	2	0	0
@	350	0	0	5	50	load	1	2	0	5
$	350	0	0	5	50	clock_tick	0	0	0	0
$	350	0	0	5	50	unload	1	3	0	0
@	350	0	0	5	50	load	1	3	0	2
$	350	0	0	5	50	unload	1	1	0	0
@	350	0	0	5	50	load	1	1	0	1
$	352	0	0	5	52	unload	3	1	0	0
@	352	0	0	5	52	load	3	1	0	4
$	354	0	0	5	54	unload	1	4	0	0
@	354	0	0	5	54	load	1	4	0	2
$	356	0	0	5	56	unload	2	1	0	0
@	356	0	0	5	56	load	2	1	0	3
$	358	0	0	5	58	unload	1	5	0	0
@	358	0	0	5	58	load	1	5	0	5
$	358	0	0	5	58	unload	3	3	0	0
@	358	0	0	5	58	load	3	3	0	1
@	402	0	0	6	42	load	2	3	0	4
@	403	0	0	6	43	load	2	5	0	1
@	553	0	0	9	13	load	2	4	0	5

INTERACTIVE PRODUCTION SCHEDULER

by

MURALIDHAR THEEGALA

B.Tech (Mechanical Engineering)

Indian Institute of Technology

Madras, India, 1987

AN ABSTRACT OF A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1989

ABSTRACT

The objective of this work is to develop an interactive simulator which will help the user in the simulation training of production management and to facilitate instruction in production scheduling by aiding the visualization of the effects of changes in a production and inventory system.

The simulator which runs on an IBM-PC or compatible computer, features a character output in addition to other features such as zoom, future event chain display, and statistics display. It is completely menu-driven and can run in automatic as well as manual mode. Typically it simulates a job shop atmosphere wherein the primary events are purchase, setup, load, unload and transfer. Scheduling of these events was made flexible by allowing the user to specify the event time in addition to the event type. It has an option to save the current simulation state on the disk and to run the simulation at a later stage from that point. Using an input file as a preload device allows one to do external analysis, say using a spreadsheet, and postulate scheduling alternatives which may be evaluated by the simulator. Finally it is interesting to contrast the simulator with the scheduling of a real-life plant whose production processes are greatly more complicated than the existing ones.